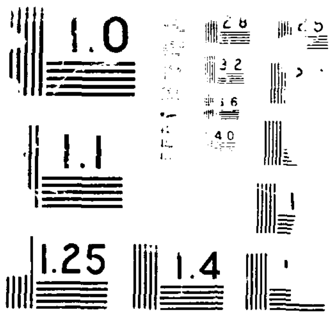
AD-A215 720 AN EXPERT SYSTEM FOR AUTOMATING [illegible]
REPLACEMENT AIRCR..(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. H D HARKEN
UNCLASSIFIED DEC 89 AFIT/GCS/ENG/89D-7 F/G 1/2 NL


1.0
2.8
2.5
3.2
3.6
4.0
2.2
2.0
1.8
1.1
1.25
1.4
1.6

(1)

An Expert System for Automating Nuclear Strike Aircraft Replacement, Aircraft Beddown, and Logistics Movement for The Theater Warfare Exercise

THESIS

Harold Dallas Harken III
Captain, USAF

AFIT/GCS/ENG/89D-7

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/89D-7

An Expert System for Automating Nuclear Strike Aircraft Replacement, Aircraft Beddown, and Logistics Movement for The Theater Warfare Exercise

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Harold Dallas Harken III, B.S.

Captain, USAF

December, 1989

Preface

The goal of this thesis was to determine a means for automating the planning section of the Air University's Theater Warfare Exercise (TWX). By delving into the areas of Artificial Intelligence and Database Management Systems, this thesis presents a flexible, efficient, and effective platform for realizing the above goal.

This thesis presents the requirements, analysis, and solutions for the realization of an automated red player for TWX. I hope I have supplied a basis on which other thesis efforts in this intriguing area of research might originate.

I am genuinely grateful to my thesis advisor, Major Mark Roth, for all those non-committal looks he presented whenever he scanned my work. Basically a pessimist, I would always try to find that little something extra which might turn that expression to perhaps a small smile or maybe even a nod of approval. It was a mischievous way of keeping me on my toes, but it worked. Thanks Mark. I also wish to thank the members of my thesis committee, Dr. Thomas Hartrum and Lieutenant Colonel Charles Bisbee for all the helpful insights they contributed and for their grammatical expertise which I always seem to lack. Finally, I would like to thank the people that I dearly missed during the time I spent struggling towards graduation; my loving wife, Whitney, and our children, Ashley and Katy. Through all the dance lessons, school picnics, and dinners at home that I missed, I am truly thankful that I still received their support, understanding, and most of all their love. Girls, Disney World here we come!

Harold Dallas Harken III

Table of Contents

Page

List of Figures

AFIT/GCS/ENG/89D-7

Abstract

The Theater War Exercise (TWX) is a five day, two sided, theater level, air-power employment decision making exercise. The decisions required are typical of those that an air component commander and staff would make. TWX is a two-sided game where the blue team is played by a student seminar and the red team is played by one or more dedicated Air Force Wargaming Center personnel who attempt to provide a realistic red opponent.

Personnel at the Air Force Wargaming Center determined that too much time was required for a red player to render an effective game. Also noted was the divergent background of the red players made it difficult to play a normalized game during multiple seminars. The goal of this thesis was to evaluate existing software programs, determine which would best serve as a platform for automating the red player, design a system to that effect, and implement it.

It was determined that an integration of artificial intelligence and relational database management systems would provide a flexible, innovative, and cost-effective approach for automation. Nexpert Object, an expert system shell by Neuron Data, was chosen as the software platform.

An object-oriented approach was used to determine the necessary structures for automating the planning section of TWX. This included the replacement of nuclear strike aircraft, the beddown of aircraft from an augmentation base, and the resolution of logistic shortfalls at each airbase due to attrition and movement of aircraft.

The creation of three knowledge bases resulted from the design phase using application prototyping, which facilitated the need for constant changes to the rules in order to present a system that acted in accordance with the desire of the red players. This new series of

programs provided a means of lessening the red player's time involved with simplistic, but time-consuming work and allowed them to increase their time on the sections dealing with target selection and prioritization.

An Expert System for Automating Nuclear Strike Aircraft Replacement, Aircraft Beddown, and Logistics Movement for The Theater Warfare Exercise

I. Introduction

1.1 Background

The Theater Warfare Exercise (TWX) is a five day, two sided, theater level, air-power employment decision making exercise. The decisions required are typical of those that an air component commander and staff would make. These decisions, once made by the exercise participants, are fed into TWX's air and land battle simulation programs, which then simulate the employment of the airpower strategy, doctrine, and warfighting principles inherent in those decisions. TWX is a two-sided game where the blue team is played by a student seminar and the red team is played by one or more dedicated Air Force Wargaming Center personnel who attempt to provide a realistic red opponent.

The requirement for TWX originated in 1976 when the USAF Chief of Staff directed the development of "...rigorous courses of study instructing operators and planners in the threat and application of force" (25:1). To accomplish this task, the Air War College conceptualized a theater level, computer-assisted wargame that would serve as the capstone for its military employment curriculum and meet the intent of the Chief of Staff's direction (22).

TWX was originally programmed to run on a Honeywell H6000 mainframe computer, but was later rehosted to a Digital Equipment Corporation (DEC) Micro Vax III microcomputer environment via the thesis endeavors of Captain Michael Brooks and Captain Mark Kross (6, 10). During this transition, TWX's structure was totally renovated from a flat file system to a more portable and flexible program using the Ingres Relational Database Management System (RDBMS). Other thesis efforts continued to improve TWX by developing a new user interface to replace the use of hard copy devices for all inputs and outputs (10, 26). A graphical interface to the wargame was introduced last year through the work of Captain Darrell Quick (16). Ongoing enhancements to TWX include porting the database to the Oracle RDBMS for use on a SUN 386i workstation.

TWX is now played extensively by the Air War College at the Air Force Wargaming Center located at Maxwell AFB, AL. The Combined Air Warfare Course, the Guard/Reserve Air Warfare Course, and the Contingency/Wartime Planning Course began utilizing the resources of TWX in 1977. TWX was also incorporated into the curriculum of the Canadian Forces Command and Staff College (1980) and the Royal Air Force Staff College (1983) as well. TWX is currently played over eighty times a year.

1.2 Problem Statement

The problem that now confronts TWX is the lack of manpower to properly supervise the overall exercise and thoroughly simulate the red player. Due to the overwhelming number of seminars run concurrently, the personnel at the Air Force Wargaming Center do not have the time to assimilate all the information given them for the next day's play. Red team players spend between five and eight hours a day inputting the next day's assignments. Specifically, the wargaming center has requested the following:

- Develop a system to free personnel from the computer interface task.
- Create a consistent and realistic opponent across all seminars.
- Provide a platform for evaluating seminars based on the strategies played by each blue team.

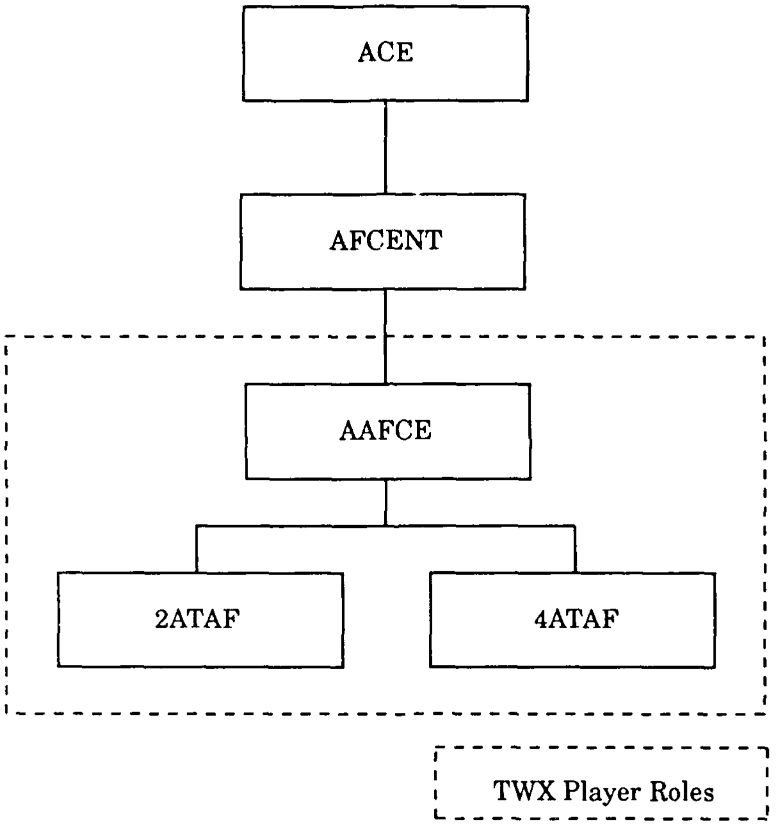
1.2.1 The Computer Interface Task. The red team makes decisions at two different levels. It should be noted that the red team uses blue terminology to represent its command structure, thus simplifying the computer-user interface. The first level of decision making is that of the Commander, Allied Air Forces Central Europe (COMAAFCE), who with his "staff" develops an air strategy to support the strategy of the theater commander, Commander in Chief, Central Europe (CINCENT), represented by the game director. The responsibilities for AAFCE have been limited to logistics management, beddown of augmentation forces, relocation of theater air forces, and rerolling of theater air forces.

The next level of decision making is at the Commander and staff of the Second and Fourth Allied Tactical Air Forces (COMTWOATAF and COMFOURATAF). At this level, players implement the Air Directives (ADs) passed down from the AAFCE commander and make decisions to ensure optimum use of their limited assets in meeting COMAAFCE's priorities and specific objectives. Figure 1 presents the organizational chart used in TWX.

TWX is still paper-intensive, utilizing massive amounts of computer printouts and a computer terminal to input user responses taken from hand-written worksheets. Players must manually review and analyze numerous computer-generated reports in order to plan their next day's strategy. Fifty to eighty percent of the red team's time is spent examining these reports and filling in spaces on a complex set of worksheets to be entered into the computer when all decisions have been made.

1.2.2 A Consistent and Realistic Opponent. There are currently twenty company grade and field grade officers serving at the Air Force Wargaming Center as red opponents for TWX. After being instructed by one of the senior players, a new member is allowed to develop his or her own strategy for successfully completing the five day seminar. This potentially allows twenty different versions of the game to be played concurrently. Thus while one blue team is thoroughly beaten, another might capture Moscow. There is a need for a consistent player

Central Region Command & Control



AAFCE - Allied Air Forces, Central Europe

ATAF - Allied Tactical Air Forces

Figure 1. TWX Organizational Chart for Blue and Red Players

whose red strategy is based on known Soviet tactics and doctrine and is not biased by the training and culture of the human opponent.

1.2.3 A Platform for Evaluating Seminars. At the end of a five day seminar, the red teams report to the blue teams they played against. The blue teams are then briefed on how well or how badly they played against *their* red opponent. Unfortunately, there is presently no way to grade the blue teams against each other since they were not exposed to the same red strategies. A platform that can evaluate blue strategies would help resolve this problem. Given a blue strategy, numerous red game plans could be tested in order to find which produced the best results. That plan could then be used against the blue team. Conversely, multiple blue strategies could be played against the same red strategy and graded according to how well they met their objectives. Students could then see which team was best prepared to meet the red strategy presented.

1.3 Proposed Solutions

Automating the red player from simply a software point of view has many obstacles.

1. The are many solutions. It would take too long to examine each one.
2. The problem solving expertise is conceptual and cannot be reduced to "numbers".
3. The information needed is incomplete, uncertain, subjective, inconsistent, and subject to change.
4. The conclusion reached will often be uncertain.
5. Experts may disagree on how to solve the problem.
6. The task is always changing and evolving (24).

The above problems tend to point out that a conventional software approach is not advisable, but that a system built using artificial intelligence (AI) might be a better one. Expert systems programs emulate the problem-solving processes of human experts through the use of AI techniques.

The use of expert system shells requires that key knowledge concepts and problem solving strategies are identified by one or more experts from the field. Red players from the Air Force Wargaming Center were interviewed in December 1988, since they were acknowledged as the known experts for planning and executing red strategy for TWX. From their ideas, a basic requirement was derived. All players wanted to see the AAFCE phase of TWX fully automated, but wanted the ATAF phase only partially automated.

Due to the complexity of a database/expert system link the scope of the initial problems was narrowed to meet the demand of the red players: fully automate the AAFCE phase of TWX.

1.3.1 Automating the AAFCE Phase. The general order of events for AAFCE is to

- collect statistics from the computer-generated reports
- maintain the strike generation level
- move aircraft in from the staging base
- rerole certain aircraft (if desired)
- ensure enough logistics are present at the airfields to accomplish the mission (23:3.6).

Since the computer has instant access to all reports there is no need to externally generate hardcopies. All information can be maintain within the system and called upon by the expert system through an interface with the database. The following is a priority list, ordered by Wargaming Center personnel, for automating the above procedures:

1. Automate rerole of aircraft to maintain 15 strike aircraft per nuclear strike base.
2. Automate beddown of aircraft from staging base, taking into consideration: base damage, shelters available, revetments available, type of aircraft already stationed on base, and amount of available ramp space.
3. Automate resolving logistics shortfalls due to enemy attacks and aircraft arrivals.

It is the objective of this thesis to provide a vehicle for the red player experts to input declarative goals and strategies that will achieve the requirements of the list above.

1.3.2 Automating the ATAF Phase. Targeting occurs during the ATAF planning portion of TWX. Red players currently make target selections from a given priority list. After targets are chosen, aircraft must be selected according to available types and missions needed to be flown. Player time is constrained due to the numerous factors involved in making aircraft selections. This results in not enough time being spent on following a realistic red strategy. Meeting the following two objectives will greatly increase the time that can be utilized in selecting proper targets and thus producing a more effective and realistic red opponent.

1. Allow the red player to select mission targets. Generate the necessary aircraft sorties needed to assure a successful mission. This includes reconnaissance, defense suppression, and electronic measures support.
2. Allow the red player to change aircraft selection when desired.

The realization of the above objectives has been assigned to a follow on thesis effort by Captain Karl Kabanek.

1.4 Assumptions

The following assumptions were made concerning the work within this thesis:

- The Air Force Wargaming Center is satisfied with the current structure of the database which resulted from the rehosting work of the TWX system.
- The Air Force Wargaming Center does not want a fully automated player for the TWX, but requires a system that allows personnel to apply their time to more importants task such as target selection.
- The Air Force Wargaming Center requires a system that is highly portable and flexible, due to the computer hardware changes presently occurring at the center.
- The decisions made by the new system must be readily verifiable.

1.5 Approach/Methodology

The approach taken for providing the above solutions is simply: learn the system, analyze the requirements, design the new system, and implement the new system. In the thesis proposal the following objectives were identified:

- Learn how to play TWX. Determine how red strategy is realized by interviewing senior red players at the Air Force Wargaming Center and observing and questioning the red play.
- Evaluate existing AI expert system shells in order to find one that provides an interface to TWX's database, operates on the hardware platform that TWX is currently running, and meets all functional requirements outlined by the first objective. An indepth discussion on this objective can be found in chapter III.
- Design an automated red player with interjection by Air Force Wargaming Center personnel at points determined by requests from the red players. The scope of the design depends on the complexity of the rules needed to generate a realistic player

and the complexity of implementing those rules. This will be further discussed in the design-oriented chapters.

- Implement the design. This is encompasses testing and validation.
- Document the system with a user's manual and system integrator's handbook, containing maintenance and installation procedures. These documents can be found in appendix A and appendix B respectively.

During the literature review for this thesis, it was noted that only a general methodology such as rapid prototyping, was advocated for designing a rule base system. Examination of a previous thesis effort on the TWX user interface (26), showed that application prototyping for software requirements analysis mapped very nicely into the methodology required for this thesis. The following is a list of requisites for application prototyping and a brief explanation why these assumptions are valid for this thesis.

1. All prerequisites are prespecified: Discussions with the Air Force Wargaming Center provided general directions of what work needed to be accomplished. However detailed requirements were not available. Also systems involving rule bases are never complete when a requirement is first derived. Most rule bases evolve after hours of interviewing experts and evaluating notes taken from those interviews.
2. Inherent communication gap: Communication of detailed requirements was hampered by a lack of understanding by the user of the expert system shell's development system and its capabilities.
3. Availability of tools for quick building: Both the Ingres' and Oracle's database development systems and Nexpert Object's Expert System Shell development system provide the necessary tools for rapid prototyping. This is a must for knowledge based systems,

since a set of rules must be tested constantly to monitor how well those rules emulate a human expert.

4. Active system required: The resulting expert system will be interactive with TWX operators.

5. Rigorous approach is correct once requirements are known: Other more rigorous approaches are applicable in different phases of the expert system development cycle, such as functional decomposition and object-oriented design.

6. Extensive iterations necessary: New problems and decision changes always arise when working with more than one expert on designing one set of rules for an expert system (4).

Determining the suitability of application prototyping for this thesis was based on evaluation of a number of factors. The following is a list of the factors and description of a type of system which is appropriate for application prototyping.

1. System Structure: Interactive and large amounts of database transaction processing.
2. Logic Structure: Very structured components.
3. User Characteristics: Uncertain about detailed requirements.
4. Application Constraints: Development time available to perform iterations.
5. Project Management: Confidence in the development system to perform application prototyping.
6. Project Environment: Prespecification difficult and capabilities unknown (4, 26).

Based on the above factors, the identified problems were good candidates for this methodology.

1.6 Materials and Equipment

The equipment used for this thesis included one SUN 386i Workstation, one Zenith 386 microcomputer, the Ingres RDBMS software package, the Oracle RDBMS software package, the Nexpert Object AI software package, and other software development tools. All equipment listed was provided by the Air Force Wargaming Center. Source code and documentation from previous thesis efforts were instrumental in understanding the current wargame implementation and integrating the new system.

1.7 Sequence of Presentation

Chapter II is a detailed literature review of the current technology for integrating artificial intelligence and database management systems. This area of research was fundamental in fulfilling the objectives of this thesis, since all data for TWX was stored in a relational database and a necessary platform had to be found in order to integrate the database with an AI development tool. Chapter III discusses the evaluation of AI expert system shells and the criteria used to make a final selection.

Chapters IV-VI detail the analysis, design, and solution to each problem presented in this chapter: automating the 3 major components of the AAFCE phase. Each chapter begins with an introduction of the actual task involved.

Next the problem is analyzed using the software engineering principles for software requirement analysis. The third section of these chapters discusses the solution to automating each task. The discussion will include the software tools used and problems encountered.

The fourth and final section of each chapter will contain a summary and any recommendations for further work in the area.

Chapter VII completes the thesis with an overall conclusion and recommendations for further development of the artificial intelligence/relational database management system link in the TWX system.

II. Literature Review

2.1 Introduction

The need for integrating Artificial Intelligence (AI) and database systems has been evident since the two areas' very beginnings. Both AI and database systems need to manage, access, and reason about large amounts of possibly shared information (13:1). AI's overlap with the database field is the knowledge-base which contains a system's inferred knowledge about a closed-world system. The key difference between knowledge-bases and other database systems is the use of semantics. (See overview of AI below.) Databases can serve as the virtual memory of an AI system, storing facts and reasoning states, while the knowledge-base can contain rules and control the focus of attention. Database systems require a database manager to provide an efficient and convenient interface between low-level data stored in the database and the application programs and queries submitted to the system. The tasks required of the database manager map well into the domain of AI where an AI system can be used as means of performing reasoning, filtering, and other tasks dealing with queries.

The Theater Warfare Exercise involves the tracking and maintenance of numerous aircraft as well as the bases within the theaters responsible for those aircraft. A relational database was used as a storage facility for maintaining the structure of the data necessary to conduct the computer exercise. A vehicle to automate portions of this exercise requires the integration of database internals to their counterparts in an AI oriented program. This chapter gives brief overviews of AI and database systems. It then discusses the terminology and three applications that illustrate key areas of integration within the two fields. Future research projects are suggested based upon research literature.

2.1.1 An Overview of Artificial Intelligence (AI). A generally accepted definition of AI is as follows:

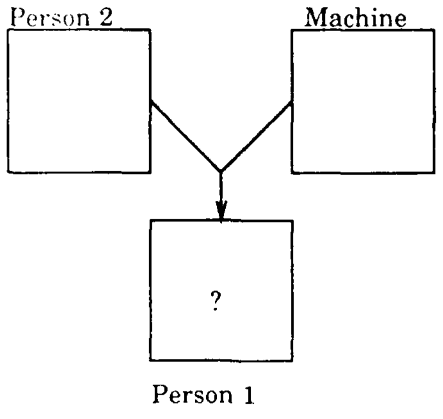


Figure 2. The Turing Test for AI

> Artificial Intelligence is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics that we associate with intelligence in human behavior – understanding language, learning, reasoning, solving problems, and so on. (2:3)

Another approach to AI was proposed by Alan Turing in 1951 (21). The now famous "Turing Test" provides a means of measuring a machine's intelligence by placing the machine and two people in separate rooms. One of the persons will then present a question to the machine and the other human being. If the person asking the question cannot distinguish between the computer's and the other person's answer, then the machine is acknowledged as having artificial intelligence. See Figure 2.

Inference is one of the major keys to AI. It is the process of creating explicit representations of knowledge from implicit ones. Most cases involving AI assume that the knowledge contained in their respective databases and any knowledge inferred by that data represents all the known information about the system. This is a closed-world system. If the information about a system does not exist in a database or it cannot be inferred then it simply does not exist. This assumption can be quite disasterous when applied to the wrong kind of problem. Therefore AI systems such as Rule Based Expert Systems must be limited to those applications that can be mapped to a closed-world system.

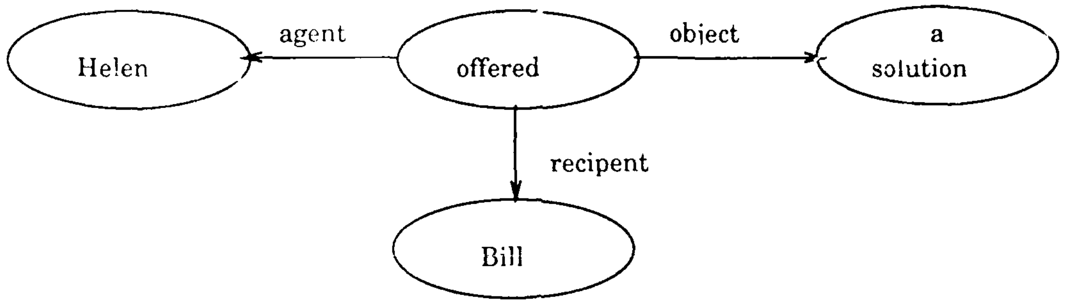


Figure 3. A semantic net for "Helen offered Bill a solution."

A promising field in AI is semantic networks. Semantic networks were first developed to represent the grammatic structure within sentences in terms of objects and their relationships. This object oriented approach is desirable since it is more efficient to represent each object once and use cross-pointers rather than duplicate the object explicitly every time it is involved within a relation. In theory semantic networks are as powerful as the predicates in predicate calculus. Unfortunately the primary use of semantic networks is providing a graphical depiction of knowledge and not an actual implementation.

A semantic network is a labeled direct graph where both nodes and edges may be labeled. There are four types of nodes: concepts, events, characteristics, and value-nodes. Concepts are the essential parameters of a modeled world and relate to physical or abstract objects. Events are used to represent actions within a world. Characteristics are used to represent states or to modify concepts, events, or other characteristics. A characteristic is similar to binary relation mapping nodes to which a characteristic may apply to a range of values that a characteristic may take. Value nodes represent the values that characteristics may take. Figure 3 shows a typical semantic net from (20:117). Other shortfalls concerning semantic networks are that a standard does not exist and reasoning methods are not provided. A database approach using semantics will be discussed in section 2.2.

Frames are another key structure in AI. A frame is a collection of knowledge relevant to a particular object, situation, or concept. Generally there are many pieces to a frame and many frames to a knowledge-base. A frame provides representation of an object in terms of a set of attribute names and values for the attributes. A frame is somewhat analogous to a "record" data type in PASCAL or 'C'.

When surveying AI knowledge representations, relational databases are considered to be highly efficient in handling data. Relational databases provide useful transformations such as selection, projection, and joins. These operations may be used in connection with more powerful inference methods (such as resolution in predicate calculus) to attain a combination of intelligence and efficiency in a knowledge-based system (20:124–127).

2.1.2 A Brief Overview of Database Systems. There are three classic models in the database arena:

- The Hierarchial Model
- The Network Model
- The Relational Model

The hierarchial and network models are the elders and are tied more closely to the underlying implementation of the database than is the relational model. These are a few reasons why the relational model is now the fastest growing commercial model of databases. Over 300 relational DBMSs are now being sold for virtually any type of hardware platform. The relational database relies heavily on its management subfunctions for interaction with its secondary storage, integrity enforcement, security enforcement, backup, recovery, and concurrency control.

The relational database consists of tables which represent a relationship among a set of values. These values or attributes together represent a unique relationship within the

Ship-Id	*Ship-Name*	*Ship-Captain*	*No-of-Crew*
NCC–1701	Enterprise	Kirk	305
NCC–1704	Constellation	Patrick	305
NCC–1706	Intrepid	Riley	325

Figure 4. Relational Database Table

database. These tables map very nicely into AI structures, facilitating the transfer of data from a database to a program that can apply AI reasoning methods. A sample table is shown in Figure 4.

Another aspect of relational databases that must be acknowledged is query languages. These procedural or nonprocedural languages are the keys to a database's internal structures, allowing users to retrieve, modify, and store data. Structure Query Language (SQL) was developed as a query language for System R (1). SQL consists of three major clauses: select, from, and where. The select clause is used to list the attributes desired from the result of a query. The from clause is a list of a relations to scanned during the execution of a query. The where clauses is the selection criterion upon which the query applies itself. SQL or one of its contemporaries today plays an important role in connecting an AI program such as an expert system to a database.

There have been many proposals and implemented systems for coupling a logic programming language (such as Prolog) with a relational database. These systems are broken into two categories:

- Loosely-Coupled Systems
- Tightly-Coupled Systems

Loosely-coupled systems regard the external DBMS and the logic programming language as communicating through an interface. For example a rule may be compiled into a relational algebraic program defining a view. A goal in the logic program triggers retrievals from the DBMS. In these loosely-coupled systems, the granularity and efficiency of the spanning interface is crucial to performance. Loosely-coupled systems are comparable to what has been successfully done with 'C' coupled to Ingres.

Tightly-coupled systems make little or no distinction between the logic programming language and the DBMS. Two basic strategies have been advocated: either extending the logic programming system to provide features such as security, data integrity, user sharing, concurrency, and backup and recovery.; or extending the DBMS to handle logic variable, structures, and deduction (14:108). However, both strategies have been found to be extremely difficult and many companies have simply decided to build loosely-coupled systems.

2.2 *Applications*

The number of applications integrating AI and database systems is rapidly growing. Since 1975 the interaction between the two areas has broadened and become more systematic (5:12). Numerous workshops, symposia, discussions, and survey papers have addressed the need for more research into the combined fields. In 1983 a survey by Jonathan King published in the SIGART newsletter listed over 30 research projects focusing on AI and database system interaction. This chapter looks at three primary areas and their impact on the the two fields they are bringing together.

Section 2.2.1 discusses the work presented by Nicholas Roussopoulos and John Mylopoulos at the first Very Large Database (VLDB) Conference. The paper proposed using semantic networks for conceptual descriptions of the contents of a database. It is one of the first research endeavors to advocate the wholesale use of AI techniques in a database management system. Section 2.2.2 explains the implementation of the knowledge-base, Krypton. An example is provided to help illustrate how it works. Section 2.2.3 reviews the internal components of an expert system and briefly discusses a new expert system shell called NEXPERT Object.

2.2.1 Semantic Networks for Database Management. The usefulness of Database Management Systems (DBMSs) is severely restricted by their failure to take into account the semantics of databases (17:112). Some of the more specific problems are listed below:

(1)*What do attributes and relations mean?* In order to use a relation or attribute a user must know what they mean.

(2)*How do we choose a relational schema for a particular database?* The concept of functional dependency is not adequate enough for expressing semantic relationships that exist between items that make up a database.

(3)*When do database operations make sense?* There are many semantic pointers that can be used to decide whether or not an operation makes sense. This expands operational control beyond simple cost and security constraints.

(4)*How do we maintain database consistency?* With the semantics of the database excluded from the relational model the effect insertions, deletions, and updates have on the database is only understood by the user in terms his/her subjective view of what the information in the database means. Thus consistency becomes a subjective notion and this can easily lead to its violation (17:134-136).

2.2.1.1 Semantic Network Integration. The best way to understand semantic integration within a database system may be through an example. Assume two companies; one company makes a part that is used by the other company. The following diagram shows a semantic network depicting the above relationship. When describing certain characteristics such as the price of a part, a "with-respect-to (wrt)" edge is used to show mappings from a cross-product domain to a range. In order to provide a price for part #7305 this must be mapped with a certain supply company since different companies have been know to market goods at different prices. This mapping produces a value node. See Figure 6.

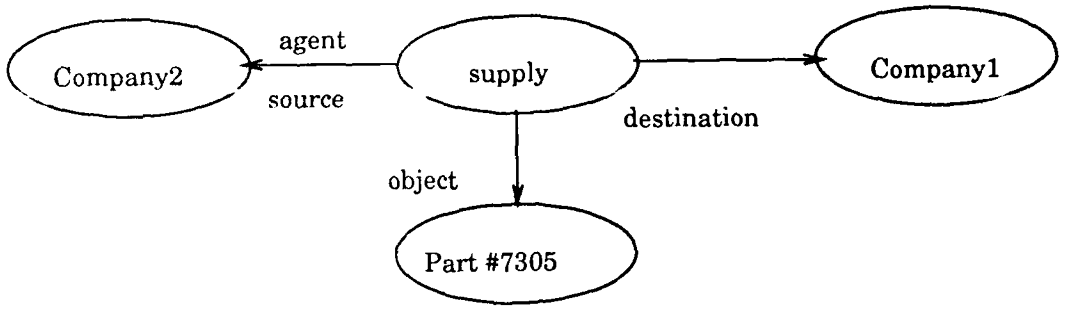


Figure 5. A semantic net for Company1 and Company2

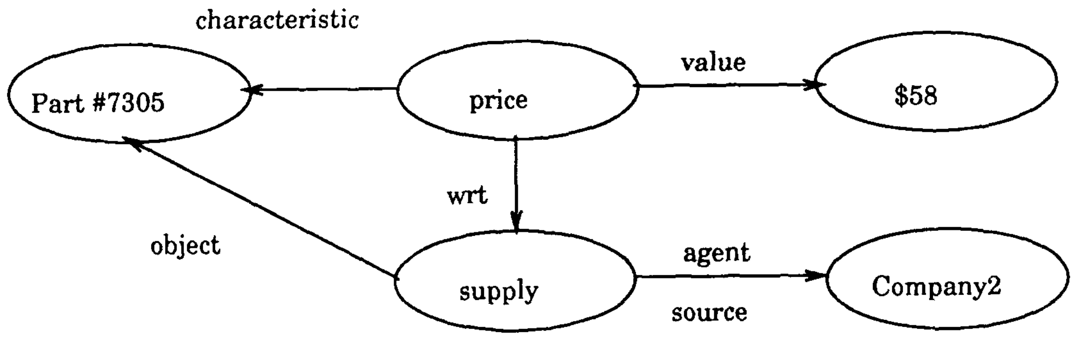


Figure 6. WRT Mapping of Company2 and Part #7305 to a price

There are four types of characteristics, depending on the relation defined between the domain and the range of the characteristic (ch - characteristic, v - value):

$$PERSON \Leftarrow ch = ADDRESS = v \Rightarrow ADDRESS.VALUE \quad \text{(Many-to-Many)}$$

$$PHYSICAL.OBJECT \Leftarrow ch = WEIGHT \rightarrow WEIGHT.VALUE \quad \text{(Many-to-One)}$$

$$PERSON \leftarrow ch - POSSESSION = v \Rightarrow PHYSICAL.OBJECT \quad \text{(One-to-Many)}$$

$$PART \leftarrow ch - PART\# - v \rightarrow PART\#.VALUE \quad \text{(One-to-One)}$$

Thus a person can have several addresses and at the same time several persons may have the same address, each physical object has a unique weight but a weight cannot be associated to a unique physical object, a physical object is possessed by a unique person, but a person does not possess a unique object, and finally, a part has a unique part number and each part number is associated to a unique part (17:115).

2.2.1.2 Semantic Operators. Semantic operators are operators that take as arguments (operands) one or more nodes of a network and construct a new node or nodes related semantically to those from whom it was obtained. Since some nodes on the net have associated relations or attributes of the database, a semantic operator may have a corresponding database operation. It is important to stress, however, that the starting point for the definition of operators is the semantic net and not the database. The following are three semantic operators that have corresponding database operators. Examples are from (17:128–132).

(1) *Selection.* The semantic operator of selection on a node n consists of creating a new subnode "below" n which has more restricted properties than node n. For example, the

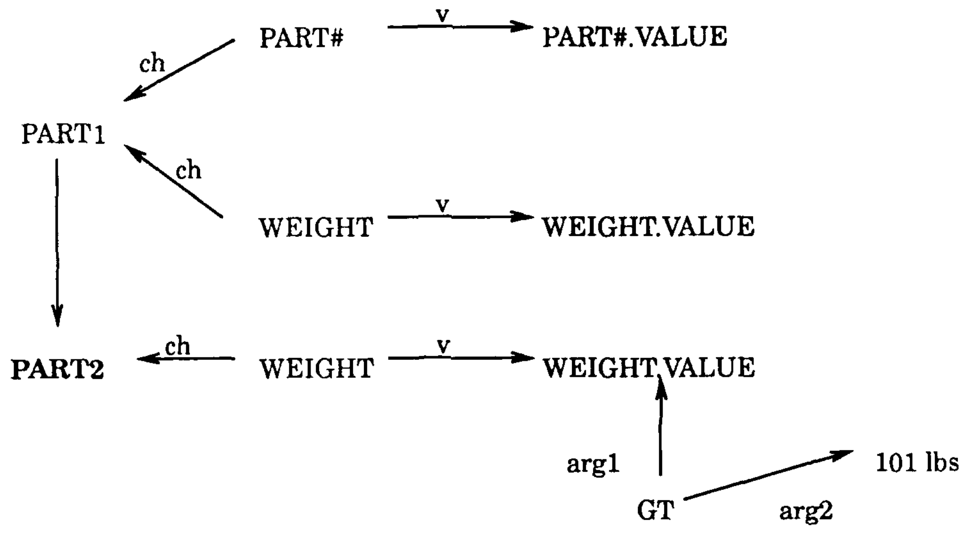


Figure 7. Selection Operation

expression 'parts *which* have a weight greater than 101 lbs.' operates on node PART1 and results in node PART2 of Figure 7.

(2) *Union*. Union operates on two nodes n1 and n2 and result in the node nr which

- is "below" every node n that is "above" n1 and n2
- is "above" n1 and n2
- inherits all common characteristics and/or cases of n1 and n2.

For example, 'cases of supplying auto.parts.made.by.ford carried out by honest.ed **or** sears with bad.boy as the destination' operates on the two SUPPLY1 and SUPPLY2 nodes in Figure 8 and results in node SUPPLY3. (Note: a - agent, d - destination, o - object, and s - source)

(3) *Intersection*. Intersection operates on two nodes n1 and n2 and results in a new node nr which

- is "above" every node that is "below" n1 and n2
- is "below" n1 and n2

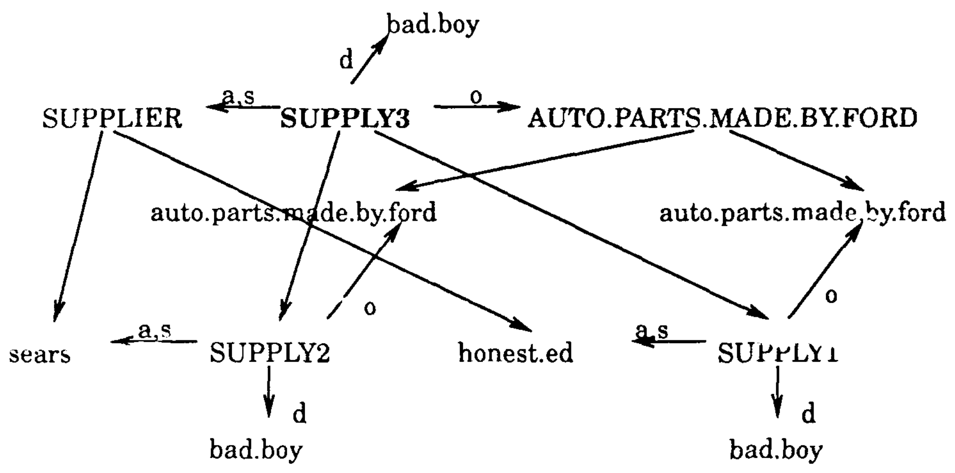


Figure 8. Union Operation

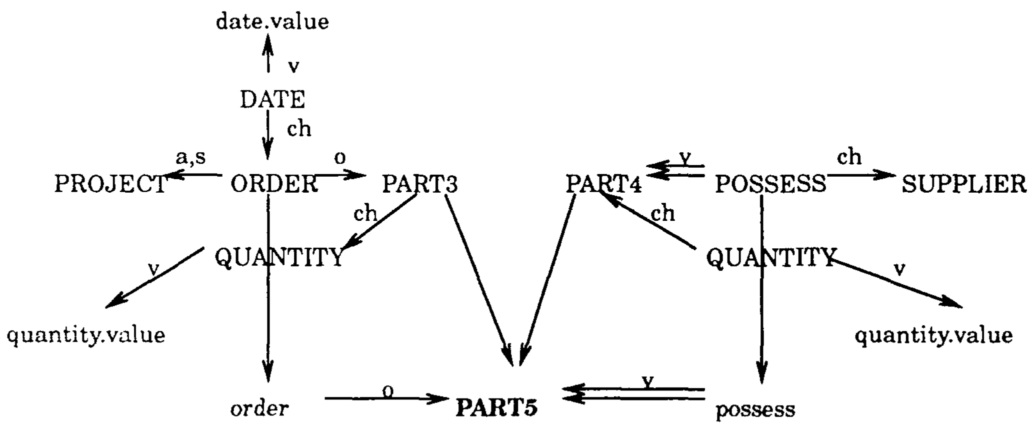


Figure 9. Intersection Operation

- inherits all characteristics and/or cases of n1 and n2.

And the last example, 'parts that have been ordered by some project **and** possessed by some supplier' operates on nodes PART3 and PART4 of Figure 9 and results in the new node PART5.

The description of the above semantic model is by no means complete. Research still needs to be done in establishing that the association of relations to the basic building blocks of the semantic net (concepts, events, and characteristics) is adequate, that the set of semantic operators proposed is in fact sufficient, and that consistency, integrity, cost and security constraints have been met. Where this model might fall short in accomplishing these goals, it sets a very nice foundation for solving them in the near future.

2.2.2 Knowledge-Bases. Several database models allow the expressing of simple facts such as "Smith has an account at the Centerville Branch." However, we are not able to make use of more complicated facts or rules such as:

- All accounts are either passbook saving accounts, checking accounts, or money market accounts.
- Passbook saving accounts pay 5 percent interest.
- Checking accounts have a \$5 per month fee.
- Checking accounts pay 5 percent interest if the monthly balance is over \$1000; otherwise they pay no interest.
- Money market accounts pay 8 percent interest if the balance is over \$2500; otherwise they pay 6 percent interest.

Rules such as these may be used for consistency constraint by transactions in the database, but in general they are not used by the DBMS to speed up queries. In fact they may never explicitly be defined within the database.

Consider the query "Find all money market accounts that pay 15 percent interest." If the system could use the fact that all money market accounts pay 8 percent interest, the system could conclude that the answer to the query is the empty set without ever accessing the database (9:474).

Rules are important since they can be used to answer queries that cannot be expressed in standard database queries. In regular databases only information about facts can be accessed and manipulated. The key to knowledge-bases is that they may be queried to obtain meta-data or data about data.

2.2.2.1 Knowledge-Base Architecture. A knowledge-base (KB) consists of two parts:

- A set of rules
- A collection of data or facts

The "collection of data" is actually a small database and like most databases it must have a manag ment system. Thus there is need for a knowledge-base management system (KBMS). The KBMS's primary goal is to "manage" the knowledge resources of a collection of KB applications (e.g., all those of an organization). It also uses unified control schemes for consistency, semantics, and knowledge content (ie. what knowledge resources the KBMS has) as well as checks for redundancy, reliability, and security (12:37). The KBMS is assumed to actively cooperate in the problem solving process.

Early KBs were sufficiently small to fit within a system's main memory and performance was not a main concern. However, the need for more sophisticated KBs has arisen in the last few years. Many new requirements have been place upon KBs and their management systems:

(1) *Large Knowledge-Bases.* KBMSs must acknowledge that they must deal with a large amount of facts in order to model the real world. Also knowledge for individual components cannot always be formulated concisely (e.g., a small set of rules). It is quite possible that a KBMSs will have to manage more that one set of KB components. This would result in large "central" KBs.

(2) *Heterogeneous Knowledge-Bases.* In typical DBMSs interfaces for multiple programming languages, query languages, report writers, etc. are needed. KBs are being developed to provide access to multiple-knowledge-representation languages and systems.

(3) *Knowledge Sharing.* Knowledge sharing is necessary for a query optimizer to plan access strategy or prestage data the user is likely to ask for next.

(4) *Multiple Data Types.* Processing of normal formatted types as well as spatial data, imagery, signals, etc. is now required by KBs. The KBMS must now handle different types of processors and their associated storage devices to retrieve and store these complex data types.

(5) *Communication Between Components.* There must be a flexible and efficient communication facility to allow the necessary flow of information between rules and data.

(6) *Integrated I/O.* Effective presentation is required of information by the system as a whole. It may be necessary to present results from several KB components at the same time. The above constraints can be related to input as well.

(7) *System Modularity.* A KB is naturally going to grow with the addition of new components, data types, and processors. It is important that each new component not have to contain excessive information on existing components.

(8) *Self-Understanding.* Most KBs are becoming large and complex. The system should have the ability to explain the criteria for its decisions to the user in a manner that can be easily understood.

(9) *Parallelism.* Several system components may need to execute in parallel. For example, the processing of sensor data must take place in parallel, as well as the support for queries and continuous displays necessary for the smooth operation of the system.

(10) *Component Adaptability.* Some components must be specialized for particular operations. However, general-purpose components that can be used in multiple environments are more desirable.

Further discussion on the architecture of KBs can be found in (12).

2.2.2.2 Krypton – An Example Knowledge-Base. Krypton is a hybrid system with two main components, one that specializes in assertional reasoning (the *ABox*), the other in terminological reasoning (the *TBox*). Each component has its own language and its own inferencing mechanism (3:294). The *ABox* language is first order predicate calculus, while the *TBox* is a special purpose frame-based language of descriptions. The heart of Krypton is the connection between the two components: predicates used in the *ABox* are actually defined in the *TBox*. Thus all the analytic inferences computed by the frame-based *TBox* must be available for consumption by in the logic-based *ABox* (11:23).

The language currently implemented in the *TBox* has two main categories: concepts and roles, corresponding to one-place predicates and two-place predicates (binary relations) respectively. These are inter-defined by the following BNF grammar:

$$\begin{aligned}
\langle concept \rangle \quad ::= \quad & \langle 1 - predicate - symbol \rangle \\
& \mid \ (\text{ConGeneric } \langle concept \rangle_1 \ldots \langle concept \rangle_n)\ n \geq 0 \\
& \mid \ (\text{VRGeneric } \langle concept \rangle \langle role \rangle \langle concept \rangle) \\
\langle role \rangle \quad ::= \quad & \langle 2 - predicate - symbol \rangle \\
& \mid \ (\text{RoleChain } \langle role \rangle_1 \ldots \langle role \rangle_n)\ n \geq 1.
\end{aligned}$$

The *ABox* language is that of a function-free predicate calculus. The grammar is as follows:

$$
\begin{aligned}
\langle wff \rangle \quad ::= \quad & (\langle k-predicate-symbol \rangle \, \langle var \rangle_1 \ldots \langle var \rangle_k), \; k \geq 0 \\
& \mid \; (\textbf{NOT} \, \langle wff \rangle) \\
& \mid \; (\textbf{OR} \, \langle wff \rangle) \\
& \mid \; (\textbf{EXISTS} \, \langle var \rangle \, \langle wff \rangle).
\end{aligned}
$$

Note that one- and two-place predicates symbols are both terms of the *TBox* language and components of the *ABox* language. To make this intersection explicit the following terms are defined:

$$
\begin{aligned}
\langle TBox-symbol \rangle \quad & ::= \quad \langle 1-predicate-symbol \rangle \;\mid\; \langle 2-predicate-symbol \rangle \\
\langle gsymbol \rangle \quad & ::= \quad \langle k-predicate-symbol \rangle \; k \geq 0 \\
\langle gterm \rangle \quad & ::= \quad \langle gsymbol \rangle \;\mid\; \langle concept \rangle \;\mid\; \langle role \rangle
\end{aligned}
$$

So gterms, as they will be understood here, are either predicate symbols or composite *TBox* expressions and each gterm has an associated arity (1 for concepts, 2 for roles, and k for each k-place predicate symbol). One final definition describes the mapping of gsymbols to relations of the same arity over the same domain.

Let D be any set. Let E be any function from gsymbols to relations over D such that E(s) has the same arity as s. Then for any gterm e, the EXTENSION of e wrt E by

(1) The extension of any gsymbol s is E(s).

(2) the extension of (**ConGeneric** $e_1 \ldots e_k$) is the intersection of the extensions of e_i, and D if k is 0.

(3) The extension of (**VRGeneric** c_1 r c_2) is those elements x of the extension of c_1 such that ⟨x,y⟩ is in the extension of r only when y is in the extension of c_2. For example the extension of (**VRGeneric** Person Child Doctor) would be the elements of x of the extension of Person such that any y such that ⟨x,y⟩ is in the extension of Child is also in the extension Doctor; that is, the above complex term stands for those persons whose children are all doctors.

(4) The extension of (**RoleChain** $r_1 \ldots r_k$) is the relational composition of the extensions of $r_1 \ldots r_k$. For example the extension of (**RoleChain** Child Child) is the set of all pairs ⟨x,z⟩ such that for some y, ⟨x,y⟩ is in the extension of Child and ⟨y,z⟩ is also in the extension of Child; that is, the expression stands for the Grandchild relation.

To facilitate the above definitions an example is necessary. The following information is known (ie. stored in the knowledge base):

- *TBox* **Definitions**:

  Primitive Roles: Child

  Primitive Concepts: Mammal, Thinker, Female

  Define Concepts:

  Person (**ConGeneric** Mammal, Thinker)

  NoSon (**VRGeneric** Person Child Female)

- *ABox* **Definitions**:

  Child(Fred, Pat)

  Child(Mary, Sandy)

  NoSon(Fred) ∨ NoSon(Mary)

With the facts above, we should be able to show that there is somebody in our defined world who is a Person and has a Child that is a Female, even though we don't know who that somebody is. This query is formulated using predicate calculus as $\exists x \exists y[Person(x) \wedge$

$Child(x, y) \wedge Female(y)$]. The intuition behind this proof is that if Fred and Mary both have children then at least one of them is a NoSon, and whoever is the NoSon is himself/herself a Person and has a child that is Female. That either Fred or Mary is a NoSon is insufficient, since the definition of NoSon does not require that a person have a Child, but only that if he/she has a Child, then that Child is a Female. The query above is proven true by refutation.

The proof by refutation proceeds by trying to derive a contradiction from the known facts and the negation of the theory. Lines 1-3 are the known *ABox* facts that will be used in the proof. Line 4 is the negation of the query.

1. Child(Fred, Pat)

2. Child(Mary, Sandy)

3. NoSon(Fred) ∨ NoSon(Mary)

4. $\neg Person(x) \vee \neg Child(x, y) \vee \neg Female(y)$

5. $\neg Person(Fred) \vee \neg Female(Pat)$

Normal resolution on 1 and ¬Child(Fred, Pat) in 4.

6. $\neg Person(Fred) \vee NoSon(Mary) \vee \neg Child(Fred, Pat)$

By 3, Fred is possibly a NoSon, which means that all his children are Female. Stating that Pat is not Female in 5 has the consequence that Pat cannot be Fred's Child. In other words, NoSon(Fred) in 3 and ¬Female(Pat) in 5 resolve away and leave a residue of ¬Child(Fred, Pat).

7. $\neg Person(Fred) \vee NoSon(Mary)$

Normal resolution on 1 and ¬Child(Fred, Pat) in 6.

8. NoSon(Mary)

By the definition of NoSon, if Fred is a NoSon then he must also be a Person, so ¬Person(Fred) in 7 and NoSon(Fred) in 3 are directly contradictory.

9. $\neg Child(Mary, y) \vee \neg Female(y)$

This time, if Mary is a NoSon, she must be a Person, so 8 and ¬Person(x) in 4 are directly contradictory, with Mary being substituted for x in the resolvent.

10. $\neg Child(Mary, y)$

If Mary is a NoSon (as stated in 8), any children she might have must be female. Therefore if there are no Females at all as stated in 9, then Mary must have no children. In the case the residue, ¬Child(Mary, y) was already part of the resolvent of 8 and 9, so it does not have to be added again.

11. **False.**

The final result comes from the resolution of 10 and 2. As you can see it was the result needed as stated above.

2.2.3 Expert Systems. An expert system attempts to emulate the reasoning of a human expert in some knowledge domain. It does this by using facts stored in a database and rules in a knowledge base. The rules are usually statements in logic and are expressed typically in the form of an if-then predicate, such as if person1 is the son of person2 and person3 is the son of person2 then person1 is the brother of person3. Of course this assumes there have not been any recent divorces in person2's history. A typical expert system in illustrated in Figure 10.

A frequent application of expert systems is problem diagnosis. Given a set of symptoms, the rules allow conclusions to be reached about the nature of the problem (9:475). MYCIN, a medical expert system, allows doctors to use computers as advisors in diagnosis and treatment of illnesses (18). The response an expert system gives to a user may be a question and not a fact. For example MYCIN might ask for more information about a patient such that the responses are likely to assist it in applying additional rules and thus obtaining a better diagnosis.

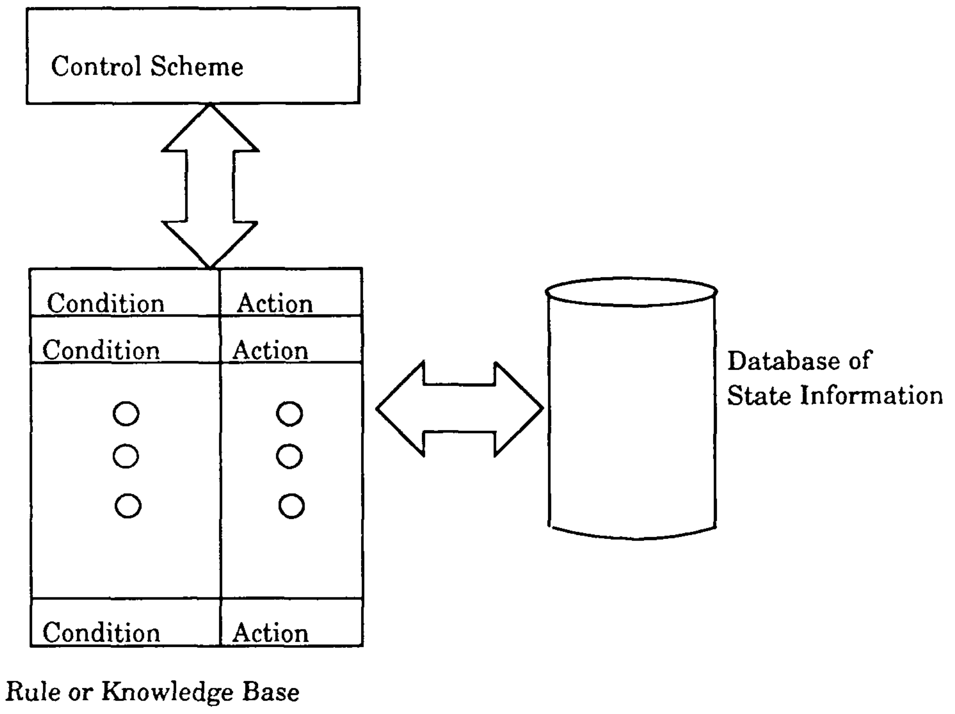


Figure 10. An Expert System

Most expert systems can find an answer to a query through the use of forward-chaining. Forward-chaining applies a given set of rules to a database; when the "if" section of the rules returns true, the "then" section is fired modifying the database accordingly. After all rules that can be invoked are used, a control procedure checks for a goal state. If a goal state is found it is returned to the user. The goal state for MYCIN would be a diagnosis to a set of symptoms. Since an expert system finds answers to a query through forward-chaining, it can explain how it reached a given conclusion by reasoning backwards. More generally it is a list of the rules that were applied in order to reach the answer or goal state. This means an expert system is not only a query processor, but it is also a collaborator.

Expert systems today use external databases for the storage of facts. This requires that the expert system submit queries in languages such as SQL and await an answer from the database system. This implementation is used not for its efficiency, but for its ease of use. It is not an optimal design since the rules in the knowledge base are not accessible for processing by the database. Also in the likely case that the expert system poses a series of related queries, the database system cannot take advantage of the similarity of the queries and must process each one individually.

2.2.3.1 NEXPERT Object – An Expert System Shell. Neuron Data's NEXPERT Object is a classic expert system shell in that it hides the underlying source code and only asks the builder to choose from the available options for inferencing methods, end-user interface, and control schemes. The builder supplies the knowledge base through the shell's interface using the shell's format for objects and rules. NEXPERT Object was developed under the premise that the domain expert should be the one directly operating on the shell without the intermediary of a knowledge engineer. To that end, emphasis has been placed on ease of use and understanding. A major asset of NEXPERT usually seen only on Lisp machines is a display of rule and object networks. The object network browser allows users to examine all the interelations between an object and the subobjects of which it is composed, as well as the properties that it can possess. Likewise, the rule network browser allows you to see every logical link between rules. The browsers are quite flexible and can display the networks either deductively (as a backward chain) or evocatively (for contextual relationships) (19).

NEXPERT Object allows applications to communicate directly and dynamically during the inference process with databases such as Oracle, Ingres, Informix, and DBASE III. Direct interfaces to these databases is integrated into NEXPERT. Users can now relate information in external databases and objects in a NEXPERT application. This is an example of a loosely-coupled system.

There is also a runtime library that allows NEXPERT to use any outside programming language to execute data manipulation. Its finest attribute is that the software created by the expert system shell can be run on virtually any type of machine without having to be edited.

This means that a product created on a SUN workstation can be run on a DEC Vaxstation, an IBM PC AT, or an IBM mainframe computer.

NEXPERT Object is used in solving a wide range of tasks such as: Classification, Troubleshooting, Maintenance, Simulation, Design, Testing, Planning, Scheduling, Intelligent Assistant, Data Structuring, Software Engineering, and many other applications. It is now used in over 60 companies and universities in the United States and in 11 other countries. It is one of the leading expert system shells available on the commercial market today.

2.3 Summary-Where Do We Go From Here?

Future computing will require the integration of many currently disjoint technologies, including AI, databases, programming languages, operating systems, heterogeneous distributed systems, and communications (7:638). AI will be necessary for handling specialized domains and for helping unique programs cooperate. Databases will be required to manage and provide access to many different types of data including rules, programs, or any other type of software object that might be created in the future.

The advance from DBMS to KBMS will probably be followed by the creation of the OSMS or Object Space Management System. Michael Brodie describes the OSMS as managing shared objects on any system in an attached network. The key objective of the OSMS is **intelligent interoperability**. Most computer systems today are disjoint such as a database and knowledge-base or they use an ad hoc interface to communicate with each other. With object-oriented approaches now becoming the main-stream methodologies of engineering, there is hope that an encapsulation of systems might be possible, thus producing general-purpose mechanisms or protocols for interoperability. The optimum use of such a connection would require tasks such as resource planning, allocation, execution, monitoring, and intervention between the two systems. This leads to the notion of a resource manager

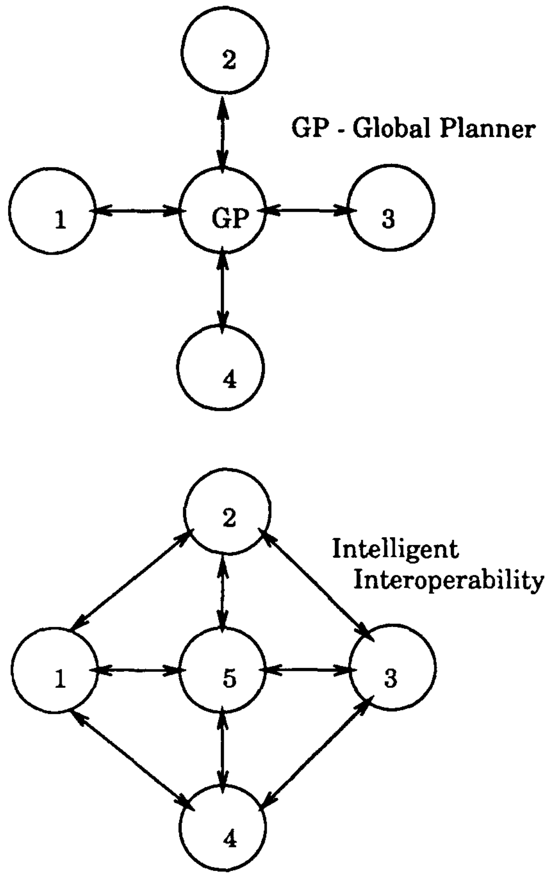


Figure 11. Global Planner vs. Intelligent Interoperability (7:639)

or global planner. The final step in intelligent interoperability would be to distribute the global planner's functions among all sharers of the network. Each user of the network would have to apply to their resource manager who in turn would find the resources necessary for execution, even if those resources were heterogeneous. Figure 11 shows the relationship between a global planner and intelligent interoperability. This vision requires the extension of database technology to general-purpose resource management and of AI technology to support distributed cooperative work. The vision relies heavily on AI and database systems integration.

This chapter has presented an overview of Artificial Intelligence and database systems integration. It has given a brief overview of both AI and database systems terms. Semantic networks and DBMS integration was discussed as well as the knowledge-base example, Krypton. Finally expert systems were exemplified by the product NEXPERT Object.

III. Evaluation and Selection of An Expert System Shell

3.1 Criteria for Selection

Before implementing any type of hardware/software combination, a best fit scenario should be given serious consideration. Unfortunately in the real world, the criteria for selecting that ideal combination is plagued with conflicts. Therefore the selection of a system is determined by a series of compromises that facilitate those ıequirements which cannot be modified or ignored.

The criteıia for selecting an expert system shell for this thesis were the following:

1. Portability of the shell
2. Representation scheme available in the shell
3. Developmental and delivery environments of the shell
4. Features available in the shell
5. Total cost of the shell

Based upon the preceding criteria, Nexpert Object by the Neuron Data Corporation was selected as the expert system shell to be used. The rest of this chapter examines in detail how Nexpert Object met or exceeded the above criteria for selection.

3.2 Portability

Ideally, a software tool should be able to satisfactorily deliver an application across the entire spectrum of hardware platforms utilized within the working environment. TWX is currently running on IBM PC compatibles and a DEC Microvax III. Future versions will run on SUN 386i workstations. Each system has a different central processing unit (CPU) and operating system to support the CPU. In order for an expert system shell to be effective in automating portions of TWX, it has to be able to port to all three hardware platforms with only minor changes at the user interface program level.

Nexpert Object is written in 'C'. This makes it portable to a wide variety of machines, including PC AT compatibles, DEC platforms running VMS or ULTRIX, SUN workstations running BSD UNIX, and Apple computers like the Macintosh. The knowledge base created by Nexpert Object is stored in an ASCII format thus allowing execution of the knowledge base on a significantly different computer by simply transferring the file.

3.3 Data Representation

Three basic inference engine types are widely used today: induction, rule, and frame. Out of the three, the rule-based system is easier to understand and therefore easier to implement. Rule based tools use a treelike representation to create symbolic structures which express deductive and evaluative progression in a reasoning path. The general format for a rule is:

if . . . **then** . . . **and do** . . .

where *if* is followed by a set of conditions, *then* by a hypothesis or goal which becomes true when the conditions are met, and *do* by a set of actions to be undertaken as a result of a positive evaluation of the rule (15:2.7).

Rules are the structures wherein reasoning takes place on a representation of the problem domain. This representation is made of interrelated objects. TWX consists of numerous objects such as aircraft and airbases on which decisions are made in order to maximize the number of fighter sorties generated. A expert system shell must be able to represent these objects and categorize them in to classes according to their shared attributes or properties. The following is a generic form for the hierarchial representation of information

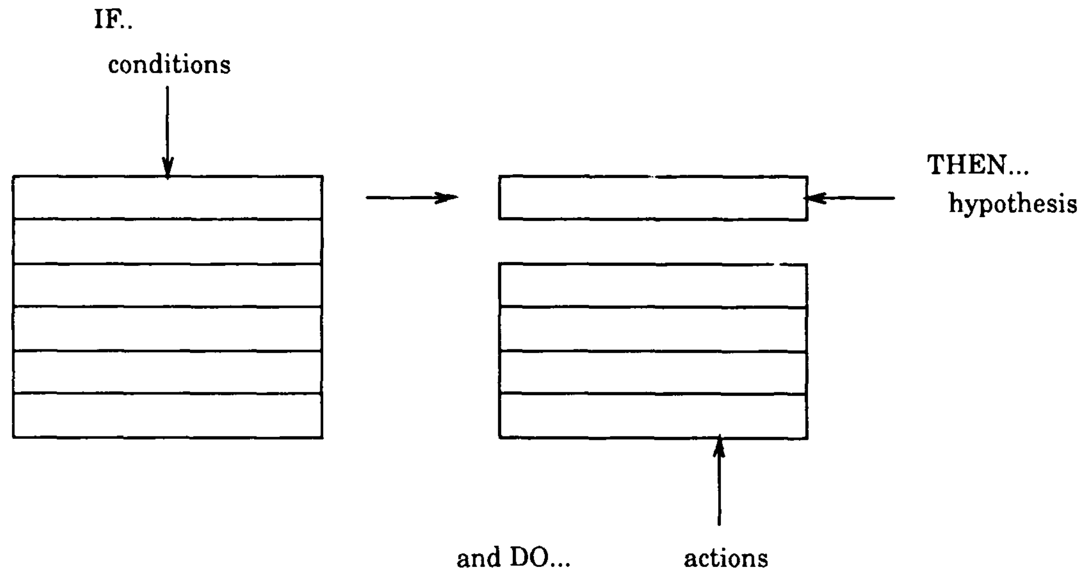


Figure 12. Nexpert Object Rule Structure

in a knowledge base:

OBJECT = Name . . . Class(es) . . . SubObject(s) . . . Properties . . . MetaSlots . . .

Nexpert Object is a hybrid system that supports both a rule-based reasoning system and a powerful object-oriented representation scheme. Rules are divided into two parts, the Left-Hand-Side (LHS) and the Right-Hand-Side (RHS). The LHS is where conditions are expressed and the RHS contains the hypothesis and actions of the rule. (See Figure 12.) Nexpert uses a hierarchial representation of domain information. Classes, objects and properties are the structures of that representation. Classes can store information relevant to all their objects and the object can inherit this information when necessary. Classes provide a way to look for objects meeting a specific condition in well-defined groups or clusters. This mechanism is called pattern-matching. Consider the following condition:

$$Is\ < AIRCRAFT > .ac\_name\ M23$$

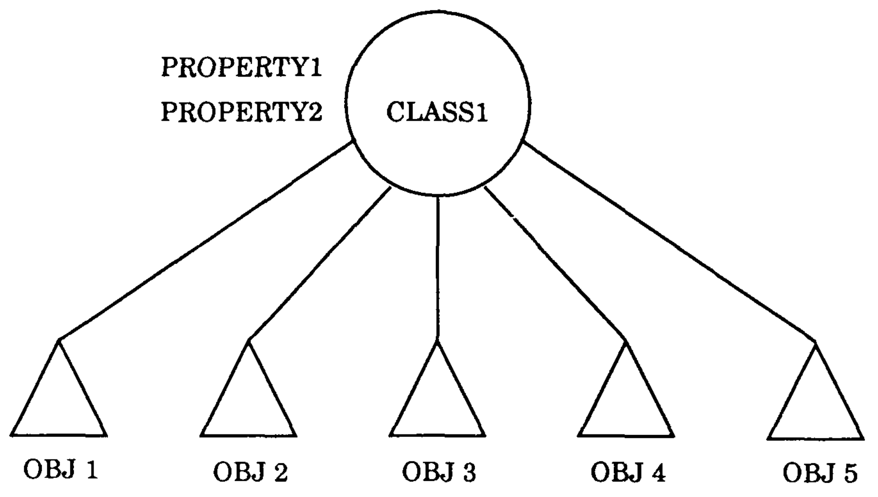


Figure 13. Nexpert Object Hierarchial Representation of Domain Information

This line translates into, "is there any airplane in the class 'AIRCRAFT' that has the name 'M23'. The brackets around AIRCRAFT denote a pattern-matching condition. The conventional way to graphically represent classes and objects are with circles and triangles. (See Figure 13.) All objects in CLASS1 have the properties, PROPERTY1 and PROPERTY2. This is an example of inheritance. Each object may have other properities.

3.4 Developmental and Delivery Environments

An expert system's inference engine may well be highly effective and efficient, but if the interface between the program and the user is not, the latter's results are visibly weakened. A shell must be able to present its output in a legible and understandable form. Entering of rules, objects, and their relations must be straightfoward and allow for modification and deletions. The delivery environment of a expert system shell is what a end user will see when an application is interfaced to the completed knowledge base. Generally this is a textual program that simply outputs the results of the inference process, but may produce graphs, informational screens, or printed output.

Nexpert Object utilizes a dynamic windowing environment for its developmental system. The actual user interface depends upon the computer system on which Nexpert resides. Nexpert Object currently uses Microsoft Windows for its IBM PC versions and X Windows for its SUN and DEC versions. The Knowledge Design Environment (KDE) for Nexpert consists of interactive knowledge agents which enable the creation, edition, modification, and display (both textual and graphical) of knowledge and its structure.

The knowledge editors which constitute the KDE are the rule editor, the context editor, the object editor, the class editor, and the property editor. Each tool is independent of the other, and can be called at any time, allowing the system to immediately take into account any modification to the knowledge base. Knowledge editors are accessed through menu bars, pop-up menus, or control-key commands. No matter where the user is in the development process, the KDE can be called up instantly (15:3.1).

Nexpert Object's delivery environment is found within its Runtime Library. This package allows the knowledge base to be accessed by external programs written in C, Pascal, Fortran, or any other type of procedural language (Embedded Coding). A knowledge base written on one machine can then be run on numerous hardware platforms different from the one in which it was developed. Nexpert can also be linked to relational databases such as Oracle and Ingres, via built in functions for database access. This is a key component in today's expert systems since intelligence requires perception and action. A knowledge-based application must be able to connect with large amounts of data to be processed and updated. Figure 14 shows how Nexpert Object links knowledge base objects and relational database tuples.

This was a high priority consideration in selecting an expert system shell since TWX data would be entered and modified directly from its database.

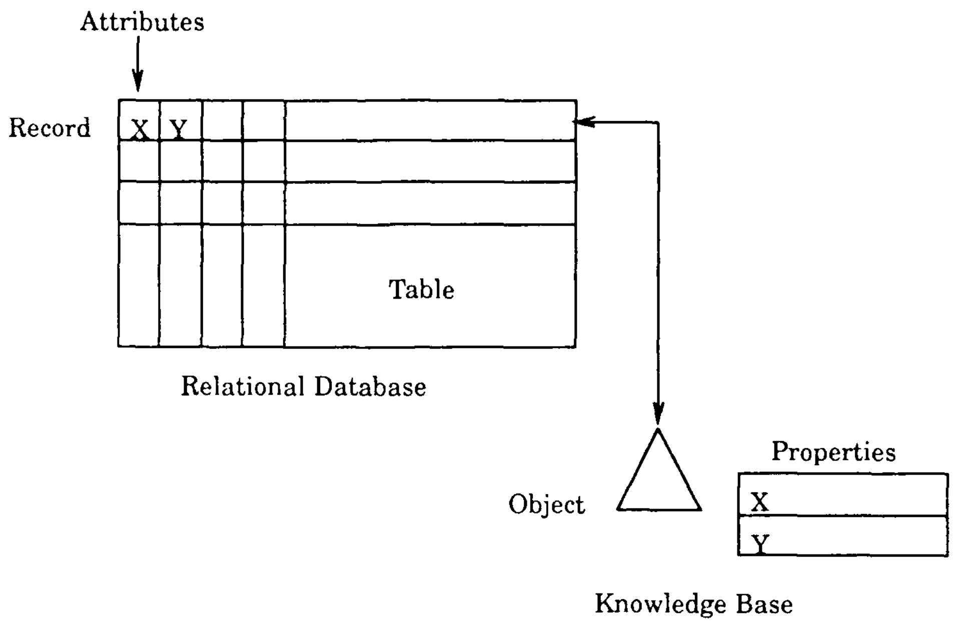


Figure 14. Nexpert Object's Knowledge Base/Relational Database Mapping

3.5 Shell Features

The way knowledge is expressed is a product of the type of shell used to represent the knowledge and the features available in the shell. It is important that a shell provide as many features as possible . These features make knowledge coding easier and provide greater flexibility. The following subsections review the features that exemplify Nexpert Object's outstanding knowledge design environment.

3.5.1 Control Schemes. The nonprocedural nature of knowledge-coding tools is both a blessing and a curse. Without an internal control language, abstract control rule structures must be used or control must be imposed via an external program. Nexpert Object allows for the above controls, but has also provided a strategy mechanism that can be used globally or on a rule-by-rule basis.

The most basic strategy modification is the control of action of effects. Whenever the value of an object, property or hypothesis is modified, the knowledge base designer must decide whether or not the system will propagate (investigate) the consequences of the modification. Nexpert Object gives the following choices:

- PWF – propagate when false
- PWT – propagate when true
- PA – propagate anyway
- PF – prcpagate forward
- EXH – exhaustive evaluation

The above strategies are boolean flags and their negations can be declared by preceding the flags with the keyword *not*. PWT propagates the inference to the next contexts encountered in the process only if the original hypothesis is true. PA propagates the inference no matter what the original state of the hypothesis was. PF will forward any RHS actions consisting in giving new or different values to data. EXH ensures that any backward chaining from a given hypothesis is exhaustive, i.e. all the rules pointing to it will be evaluated whatever the results of the previous rules.

Nexpert Object also allows dynamic modification of inheritance search routines and inheritability strategies for objects and classes.

3.5.2 Graphical Representation. A picture *can* be a worth a 1000 words ... if only you can get the picture to the screen or printer. An expert system should be able to provide a network diagram. A network diagram can be presented as either indented text or preferably, a graphic picture. The diagram helps by showing the programmer or user an overview of the structure and organization of the program's logic. The ability to see a diagram of the decision network helps you to identify missing fragments of logic and unnecessary duplication in the logic (8:147).

Nexpert Object makes full use out of its dynamic windowing system to produce the inspector program where selectivity and focus of attention are key mechanisms. The inspector program can show a complete network diagram as a meshed graph of semantically linked rules and data. The program can also localize investigations, thus allowing users not to lose their focus of attention. That is, the inspector restricts the area of interactions to a group of rules leading to a given hypothesis (semantic restriction), or to a well-defined zone of the knowledge network (spatial restriction). From this starting point, the restricted area is expanded by the user. This mechanism of selecting a knowledge area, and then expanding it, or working on it, is referred to as Navigation Investigation.

As the user's focus of attention shrinks to a smaller number of relevant concepts to investigate, he or she has the option to remove selected (either spatially or semantically) parts of the knowledge network from the display. Moreover the inspector program works in either single-focus or multi-focus mode. In single-focus mode, only one investigation is pursued. When the user re-focuses on a selected rule or data, the previous investigation is removed from the display. In multiple-focus mode, the inspector program enables any number of investigations to be concurrently performed and displayed. That is, whenever a new knowledge island is created displayed for expansion, previous investigations are not removed from the screen (15:5.1).

3.5.3 Why/How Explanation Facilities. It is important for a logic program to give some explanation of its reasoning to the user of the program. When debugging a program, there is often a need for detailed trace information which goes beyond the simple explanation facilities for a single rule. The user may be interested in the overall flow of the logic (what happened and when), not just the logic behind a single goal.

Nexpert supports three tracing utilities for the user. The transcript window provides continuous tracking of the rules currently being used by the inference engine and the data modified by the execution of those rules. The case study window displays a dynamic list of data currently known to the system with their current values, as well as the confirmed and rejected hypotheses. The final tool, called the full report window, provides the rationale behind the utilization of each and every rule applied by the system to draw its conclusions. There are also separate windows that show the current rule, the current hypothesis, and current conclusions within a system. All the above windows can be output to a printer or sent to a file for later editing.

3.6 Cost

Costs for knowledge systems are not very different from those of other, more conventional, systems. PC-based tools range from \$99 to \$10,000 (1988), minicomputer tools (specialized workstation tools fall into this category) from \$1500 to \$75,000, and mainframe tools from \$25,000 to \$250,000 (8:144). Generally, minicomputer software is 10 times more expensive than PC software. A higher price, however does not necessarily mean more functionality. It is important to match a tool's existing functions and cost to an application's needs while considering all the previous selection criteria before making a final decision. The bottom line is to pick a system that delivers the functionality to complete a project effectively and efficiently.

Nexpert Object's price tag was well below the maximum price indicated above and those of its competition. If a major hardware change occurs a small update fee will be charged in order to re-host the development system. The table in Figure 15 shows the approximate cost of the system at time of purchase.

Item Description	*Retail Price ($)*	*Education Discount Price ($)*
Development System	8,000	4,800
One Year Support	2,000	2,000
Database Bridge	1,200	720
Runtime Library	1,500	900
Shipping Costs	30	30
Total Cost	12,730	8,450

Figure 15. Purchase Costs for Nexpert Object

3.7 Summary

A number of expert system shells were evaluated with the above criteria, and all had their advantages, such as speed or low cost, as well as their disadvantages, such as non-portability or insufficient graphic representation. Nexpert Object was chosen because it best fit the requirements for this thesis.

IV. Replacement of Nuclear Strike Aircraft

4.1 Requirements

The AAFCE portion of TWX is divided into three events. The first event that must be completed is the replacement of nuclear strike aircraft at certain airbases (See Figure 16.) Executive directors require that each side must maintain aircraft capable of nuclear strike missions at all times. Personnel, acting in the role of the theater commanders, must generate reports showing the status of all bases within the red theater in order to locate which strike bases are short the number of strike aircraft required. The officers must then locate replacements for the aircraft that were destroyed. There are two means by which strike aircraft can be replaced.

- Re-role attack aircraft of the same type that already exist at the strike base
- Move attack aircraft of the same type from the augmentation base and then re-role them to strike capability.

Red experts desired a knowledge base that could decide whether or not to re-role aircraft or move new aircraft in from the augmentation base. By creating the necessary objects and rules associated with those objects, the replacement of nuclear strike aircraft could be fully automated.

4.2 Analysis

Analysis for creating the knowledge base was broken into the object-oriented process for developing classes and objects that allow data to flow between the knowledge base and the TWX database, and the process for generating the necessary rules for the knowledge base.

4.2.1 Object-Oriented Design. Both the Nexpert expert system shell and the relational database management system used by TWX provide an excellent platform for object-oriented

Strike Base	*Aircraft Type*	*Number of Aircraft*
23	U17S	15
26	M27S	15
28	M23S	15
43	M27S	15
46	U17S	15
47	M27S	15
49	U17S	15
58	U17S	15
69	M27S	15
87	U17S	15
92	M23S	15

Figure 16. Required Red Nuclear Strike Aircraft

design. Tables within the TWX database can easily be realized as classes within a knowledge base.

The tables necessary for automating nuclear strike aircraft replacement were the *rd_ac_on_ab* table, and the *rd_strk_ac* table. The *rd_ac_on_ab* table contains information on all aircraft at each red airbase such as aircraft name, aircraft role, and number of aircraft. The *rd_strk_ac* table contains the required number of nuclear strike aircraft required at a given base much like the table in Figure 16. Using the above tables, the knowledge base would have full access to the number of actual strike aircraft on an airbase as well the number required to be on base. Thus the knowledge base needed two classes in which to organize that information. Accordingly, a class was created for tracking the number of strike aircraft on base and a class for tracking the number of attack aircraft for re-role purposes were created. Since both of these classes would share properties such as airbase id's, aircraft names, and aircraft roles, it was easier to make them sub-classes of an airbase class and an aircraft class and allow them to inherit the common properties. (See Figure 17.)

4.2.2 Rule Generation. It is much easier to graph out the conditions, actions, and contexts of rules before actually writing them. Data flow diagrams, decision trees, and

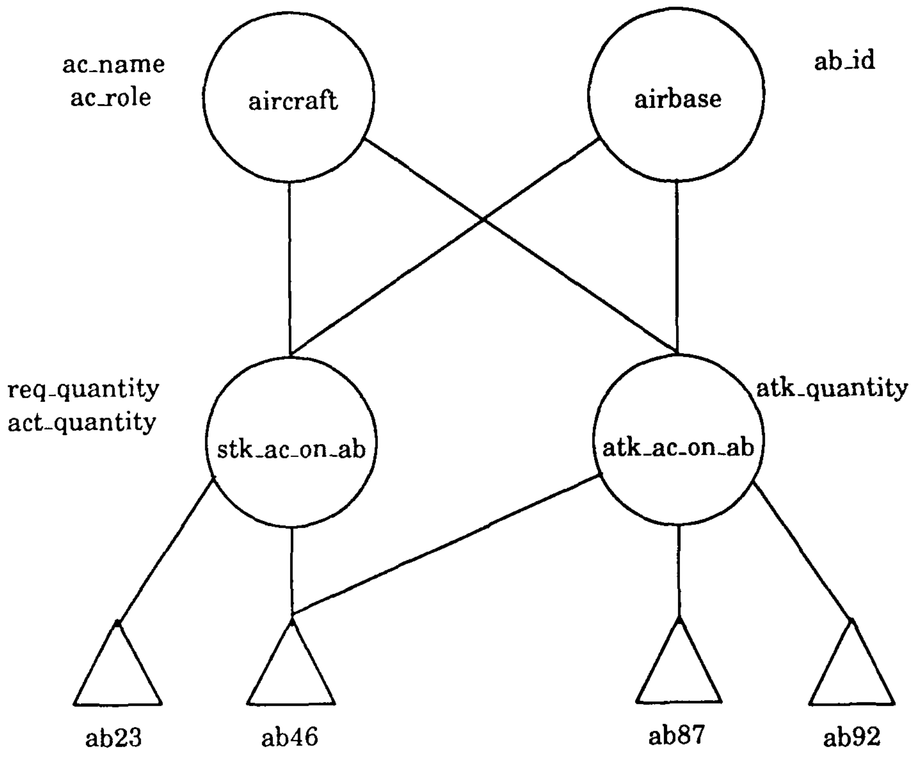


Figure 17. KB Classes for Automating Nuclear Strike Aircraft Replacement

decision tables are quite useful when creating rules necessary for an application. Due to the large number of conditions found in the later portions of the AAFCE phase and the uncomplicated depiction of conditions and actions, decision tables were used to create and display rules. Rules that do not share or directly modify data within other rules are called knowledge islands. Knowledge islands can propagate or "fire" other rules that have been placed in context with them. How rules are placed in context is explained in the solution section of this chapter. A simple flow diagram will be used to illustrate the relationship between rules, ie. their context.

Automating the replacement of nuclear strike aircraft requires the following:

1. Load the actual and required number of strike aircraft from the the database into the knowledge base.
2. If the actual number of aircraft is less than the required number then re-role aircraft of the same type stationed on the base or move in new aircraft for re-role from the augmentation base.
3. Update all database tables involved, such as the *rd_ac_on_ab* table for the airbases with strike aircraft and the augmentation base.

Figure 18 illustrates the flow of control needed for the above requirements. "F" in the figure stands for a false result but may also be used when the result of the condition is unknown such as at the start of the knowledge session. "T" depicts only true results.

The first rule in the knowledge base was created to read in the appropriate data from the TWX database. In order to start a knowledge run, either a hypothesis must be suggested or a data value volunteered. Since all hypotheses are unknown at start up, suggesting "data is loaded" would force the machine to evaluate the state of the hypothesis by investigating the conditions leading to that hypothesis. This is known as backward chaining. Rule number

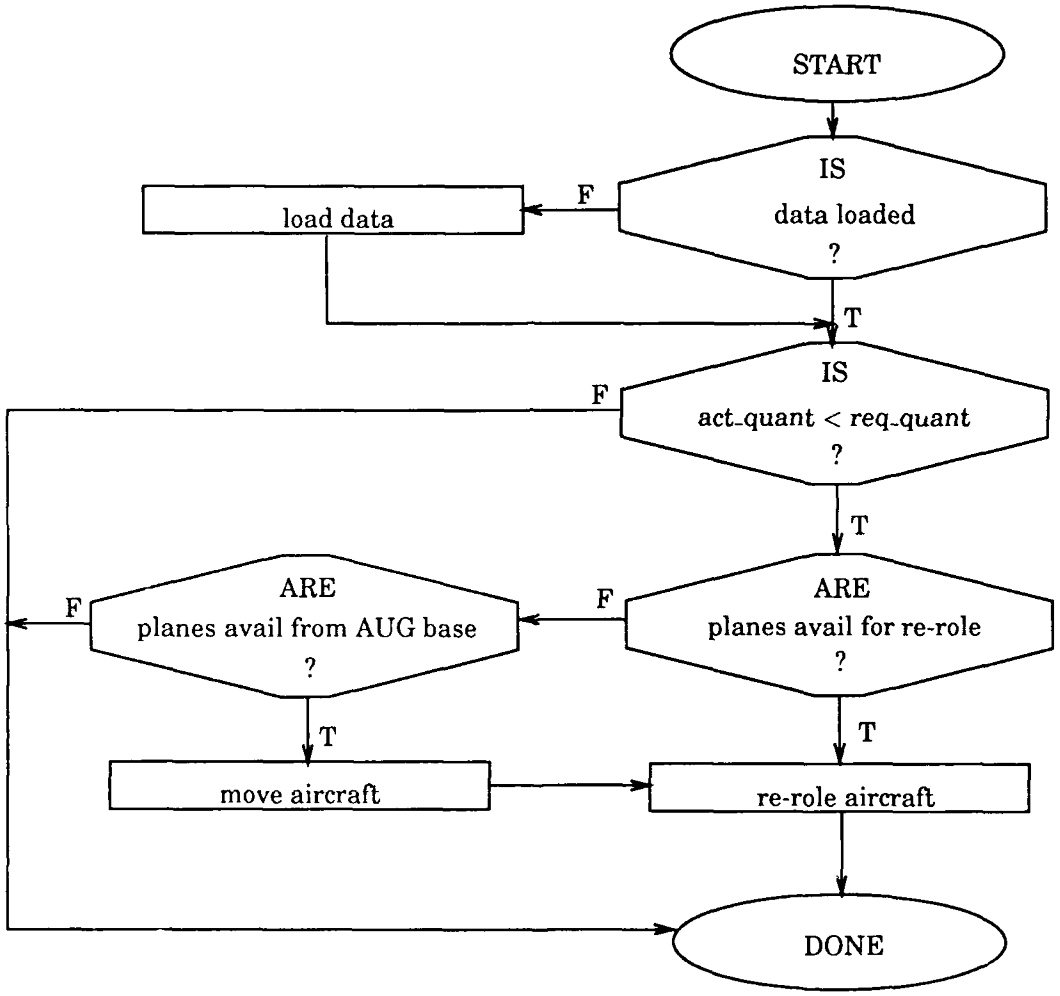


Figure 18. KB Rule Relationships for Automating Nuclear Strike Aircraft Replacement

CONDITIONS		HYPOTHESIS
READ in Actual Quantity		data_loaded
READ in Required Quantity		
		FIRE next rule
		ACTIONS

Figure 19. Decision Table for *data_loaded*

one's hypothesis was appropriately, *data_loaded*. The conditions for *data_loaded* required that data concerning the actual and required number of strike aircraft be loaded in from the database. If the hypothesis was false then an error was raised during a read from the TWX database and the knowledge session would end. If the hypothesis was true then the next rule in context with *data_loaded* would be propagated. All airbases with strike aircraft would be read into the class *stk_ac_on_ab* and identified by their airbase id number. Figure 19 shows a decision table for rule number one. In the figure conditions are shown in the left-hand side of the box. All conditions must be true in order for the hypothesis, found in the upper right-hand corner, to be true. If the hypothesis is true then the actions found on the right-hand side of the box are executed in sequential order.

Rule number two was responsible for deciding whether or not there were any airbases that had fewer than the required number of strike aircraft. If the hypothesis was false then the session was complete. If the result of the conditions was true, the bases that were low on strike aircraft would be assigned to the new class, *atk_ac_on_ab*, which would allow the number of attack aircraft at that base to be retrieved from the database. The attack aircraft and the strike aircraft would be of the same type, ie. a M27-A and a M27-S are of the same type. After the knowledge base read in the new data, the rule would fire the next set of rules that would evaluate whether there were enough aircraft on base to handle the shortage

CONDITIONS	HYPOTHESIS
Is act_quantity < req_quantity	low_on_stk_ac
	ADD to atk_ac_on_ab class
	READ in Atk Quantity
	FIRE next set of rules

ACTIONS

Figure 20. Decision Table for *low_on_stk_ac*

or would aircraft have to be moved from the augmentation base. The above hypothesis was *low_on_stk_ac*. Figure 20 shows the decision table for rule number two.

To determine if attack aircraft stationed on a base could be re-roled to their strike configuration, the knowledge base required two rules. Rule number three re-roled all attack aircraft to strike aircraft if the number needed was greater than or equal to the number of attack aircraft on base. The hypothesis for this rule was *rerole_atk_ac_from_same_base_all*. It is possible to re-role all attack aircraft at an airbase since new atttack aircraft will be moved in from the augmentation base as long as there is enough ramp space available and the base is not too severly damaged. The fourth rule re-roled only a portion of the attack aircraft if the needed number was less than the number of attack aircraft on base. The hypothesis for this rule was *rerole_atk_ac_from_same_base_some*. The actions of both rules were the same. If the hypothesis was true then the maximum number of attack aircraft needed were re-roled. If the number of attack aircraft failed to replenish the required number of strike aircraft the next set of rules would be fired in order to move aircraft in from the augmentation base. If the required number of aircraft was provided then the knowledge session would reset these two rules in order to check for shortages on other bases. Figures 21 and 22 show the decision tables for rules three and four respectively.

CONDITIONS		HYPOTHESIS
Is act_quantity < req_quantity		rerole_atk_ac_from_same_base_all
Is atk_quantity ≤ number needed		
Is atk_quantity > 0		act_quantity ⇐ act_quantity+atk_quantity
		atk_quantity ⇐ 0
		FIRE rules for moving in aircraft

ACTIONS

Figure 21. Decision Table for *rerole_atk_from_same_base_all*

CONDITIONS		HYPOTHESIS
Is act_quantity < req_quantity		rerole_atk_ac_from_same_base_some
Is atk_quantity > number needed		
		act_quantity ⇐ req_quantity
		atk_quantity ⇐
		atk_quantity-number needed
		FIRE rules to check next base

ACTIONS

Figure 22. Decision Table for *rerole_atk_from_same_base_some*

CONDITIONS	HYPOTHESIS
Is atk_quantity $\leq$ number needed	bring_atk_ac_from_aug_ab_all
Is atk_quantity = 0	
READ in quantity at augm. base	atk_quantity $\Leftarrow$ aug_quantity
Is aug_quantity $\leq$ number needed	aug_quantity $\Leftarrow$ 0
Is aug_quantity > 0	RESET & FIRE rules for re-roling aircraft

ACTIONS

Figure 23. Decision Table for *bring_atk_ac_from_aug_ab_all*

CONDITIONS	HYPOTHESIS
Is atk_quantity $\leq$ number needed	bring_atk_ac_from_aug_ab_some
Is atk_quantity = 0	
READ in quantity at augm. base	atk_quantity $\Leftarrow$ number needed
Is aug_quantity > number needed	aug_quantity $\Leftarrow$ aug_quantity-number needed
	RESET & FIRE rules for re-roling aircraft

ACTIONS

Figure 24. Decision Table for *bring_atk_ac_from_aug_ab_some*

The last two rules were created in the same manner as rules three and four. Rule number five moved all attack aircraft from the augmentation base if the number needed exceeded or equaled the quantity of aircraft on station. Rule number six moved the required number of aircraft if base supplies surpassed the needed amount. Again the action of these rules were the same except for the actual number of aircraft to be move. If either rule was found to be true then the rules necessary for re-roling the new aircraft were reset and placed on the agenda to be evaluated. The names of the hypothesis for rule five and six were *bring_atk_ac_from_aug_ab_all* and *bring_atk_ac_from_aug_ab_some* respectively. The decision tables for theses rules are in Figures 23 and 24.

4.3 Solution

Nexpert Object allowed for easy implementation of the knowledge base's classes and rules. The actual link between the TWX database and the knowledge base was accomplished using a combination of TWX database's structured query language (SQL) and Nexpert's database bridge software. The rest of this section describes the unique problems found while automating this portion of 4AFCE planning and how they were solved.

The following SQL statement generated the required number of strike aircraft and the bases at which they were stationed from the relational database.

```
select ab_id, ac_name, ac_role, quantity
from rd_strk_ac_on_ab
```

The actual number of strike aircraft on station required the joining of the two tables, *rd_strk_ac_on_ab* and *rd_ac_on_ab*. The following shows the SQL statement used:

```
select b.ab_id, b.ac_name, b.ac_role, b.quantity
from rd_strk_ac_on_ab a, rd_ac_on_ab b
where a.ab_id = b.ab_id
and   a.ac_name = b.ac_name
and   b.ac_role = "S"
```

The same SQL statement as the one used to produce the actual number of strike aircraft constructed the actual number of attack aircraft on base, except the ac_role was changed from "S" to "A". The following SQL statement generated the number of attack aircraft at the augmentation base: (Note: The augmentation base has an ab_id of 96.)

```
select a.ab_id, a.ac_name, b.quantity
from rd_strk_ac_on_ab a, rd_ac_on_ab b
where b.ab_id = 96
and   a.ac_name = b.ac_name
and   b.ac_role = "A"
```

The resulting data from the above SQL statements is loaded into the knowledge base only when called for by a read instruction within a rule.

Nexpert Object's context editor established the flow of control between rules. By placing one hypothesis in context with another, the confirmation of the first hypothesis would place the second hypothesis on the agenda to be investigated. When *data_loaded* is found to be true it must fire the rule responsible for locating strike bases with shortages. Thus *low_on_stk_ac* is placed in context with *data_loaded*. The following is a summary of contexts for the hypotheses within the knowledge base:

```
data_loaded:
    low_on_stk_ac

low_on_stk_ac:
    rerole_atk_ac_from_same_base_all
    rerole_atk_ac_from_same_base_some

rerole_atk_ac_from_same_base_some:
    rerole_atk_ac_from_same_base_some

rerole_atk_ac_from_same_base_all:
    bring_atk_ac_from_aug_ab_all
    bring_atk_ac_from_aug_ab_some

bring_atk_ac_from_aug_ab_some:
    rerole_atk_ac_from_same_base_all

bring_atk_ac_from_aug_ab_all:
    rerole_atk_ac_from_same_base_all
```

The context between two rules can also be thought of as a calling mechanism. If *low_on_stk_ac* is in context with *data_loaded* then in theory, *low_on_stk_ac* calls *data_loaded* if the hypothesis is found to be true. The strategy mechanism in Nexpert Object must be set to "Forward Confirmed Hypothesis (PWT)" to facilitate the above actions.

The use of two temporary objects accomplished updating the TWX database. *Res_temp* was created to update the strike and attack aircraft quantities after re-roling has taken place.

Aug_temp was created to update the augmentation and receiving base after aircraft were transferred between them. Nexpert Object allows special attributes called an IF-CHANGE slot for objects declared within the knowledge base. If a designated slot is changed during a knowledge session, a set of actions, separate from the rules, can be executed. The above objects contain a property called *diff.* When this property changes values, the TWX database is updated and a screen is displayed to the user describing what action the knowledge base has taken. A logfile is also updated so that a hardcopy of the knowledge base's actions can be recalled at a later time.

Variables used within the knowledge base are initialized using Nexpert's ORDER-OF-SOURCES utility. When a rule needs the value of an unknown property, Nexpert examines a table of prioritized sources where the value of the property might be found. If Nexpert cannot locate a value it simply asks the user to enter one. Since the purpose of this effort is to eliminate user interaction, we initialized all properties to zero or null using the INIT-VALUE command within the utility. This command initialized the selected properties to their suggested values upon start up of the knowledge session.

The knowledge session is started by using Nexpert's FORMS-INPUT program. This program allows a programmer to use specialized commands to create a screen-oriented interface to the knowledge base. Using FORMS-INPUT a introductory message is displayed, explaining the purpose of the knowledge base. The program then prompts the user to click on an icon marked "continue" which then loads the knowledge base, suggests the hypothesis, *data loaded*, and starts the knowledge session. The program is then used to display actions taken by the knowledge base until the session concludes. The user can then exit from Nexpert and recall the results from a log file or continue with other portions of the AAFCE phase of TWX.

4.4 Summary

In the current version of TWX, the red player is guaranteed to have enough replacement aircraft at the augmentation base to handle any shortage discovered throughout the course of the exercise. Thus, moving attack aircraft from the augmentation base should cover all replacement requirements. A suggested enhancement might be to create additional rules for the knowledge base to allow for the event that the augmentation base cannot provide the required number of attack aircraft. The rules would have to look for other airbases that did not contain strike aircraft (a red player would not want to borrow from a base that might need them later), but have the same type of aircraft in an attack configuration. The new rules could then fire the original rules responsible for re-roling the attack aircraft. This would make the knowledge base more adaptable and responsive to "real" world events.

Automating the replacement of nuclear strike aircraft using Nexpert Object was an order of magnitude easier than trying to write a program in a procedural language such as 'C' or FORTRAN. After creating the classes and objects necessary for interfacing with the TWX database, creating the flow control diagram and the rules' decision tables, Nexpert Object simplified the knowledge base construction by providing a dynamic and flexible interface for entering the above information.

V. Aircraft Beddown

5.1 Requirements

The second event required for the successful completion of the AAFCE portion of TWX is the bedding down of new aircraft from the augmentation base. For this milestone, TWX players are required to:

- find the type, role, quantity, and ramp space needed for the aircraft scheduled for relocation
- check on which bases the aircraft are allowed
- check which bases have the highest status
- check which bases have the highest number of shelters and revetments
- check which bases have the highest amount of ramp space available

Airbase status refers to the numerical value given to each airbase in order to determine its present state of readiness. A base status has a range from zero to one, with one being the highest. The number of shelters and revetments are also determined for each airbase. The only difference between shelters and revetments are that shelters provide better protection for the aircraft. The above information comes from reports that are printed out each day of the exercise.

Personnel at the Air Force Wargaming Center required a knowledge base to automate the beddown of all aircraft moved from the augmentation base. The criteria for the knowledge base was as follows:

1. Prioritize, according to player directives, the aircraft at the augmentation base for relocation

2. Prioritize the airbases according to their status, number of shelters and revetments, and ramp space available

3. Move aircraft only to airbases where they are allowed

5.2 Analysis

The analysis section is divided into three subsections. The first subsection concerns the initial development for the airbase and aircraft prioritization schemes. The second subsections contain the object-oriented design process for this knowledge base and the last subsection reviews the rule generation process.

5.2.1 Prioritization of Aircraft and Airbases. Allowing a player to prioritize the acquisition of aircraft from the augmentation base required a new attribute in the database table, *rd_aircraft*. We named the attribute "merit" and gave it a range from 0 to 100, with the highest merit equal to 100. Thus, a simple screen could be created, displaying the type, role, number, and merit of the aircraft scheduled for relocation from the augmentation base. This screen would permit the player to choose which aircraft would be moved first by assigning the selected aircraft with the highest merit. The aircraft would then be transferred according to their merit in descending order.

Prioritizing the airbases for aircraft relocation required an algorithm that would produce a base merit derived from the base status, number of shelters, number of revetments, and ramp space available. We determined that each variable in the algorithm would be mutually exclusive. This meant that if an airbase had a status of 1.00 while another base had much better shelters and more ramp space, then the airbase with the higher status would still be chosen. The reason for this decision was that the key strategy to bedding down new aircraft was to produce the maximum number of sorties that could be flown from each base. The principal element in determining sortie generation was airbase status. Each of the other elements for determining an airbase's status were also given the same treatment with the elements prioritized as follows:

1. Base status

2. Number of shelters

3. Number of revetments

4. Rampspace available

The resulting algorithm was as follows:

$$Merit = (Status * 10000 + 3) + (Shelters * 10 + 2) + (\frac{Revetments}{10} + 1) + \frac{Rampspace}{1000}$$

The divisors/multipliers used remove the order of magnitude differences between the variables, while the addition operations within the parentheses order the variables according to their priority. It should be noted that the above algorithm would have best been implemented as a set of rules. However, this required the use of an "or" condition and the procedure for implementing this condition in Nexpert would not produce the correct results.

5.2.2 Object-Oriented Design. Details needed by the knowledge base came from three tables within the TWX database. Information dealing with aircraft name, role, merit, and ramp space required by the plane came from the table, *rd_aircraft*. Data on what aircraft were available for the augmentation base came from the *rd_ac_on_ab* table. The table *rd_ac_al_on_ab* contained data on which bases a specific type of aircraft could be stationed. The knowledge base required two classes to store the above information. The class *ac_on_augm_base* contained all data from the *rd_ac_on_ab* and *rd_aircraft* tables, and the class *ac_al_on_ab* saved all data from *rd_ac_al_on_ab* where the aircraft were those at the augmentation base. Both classes

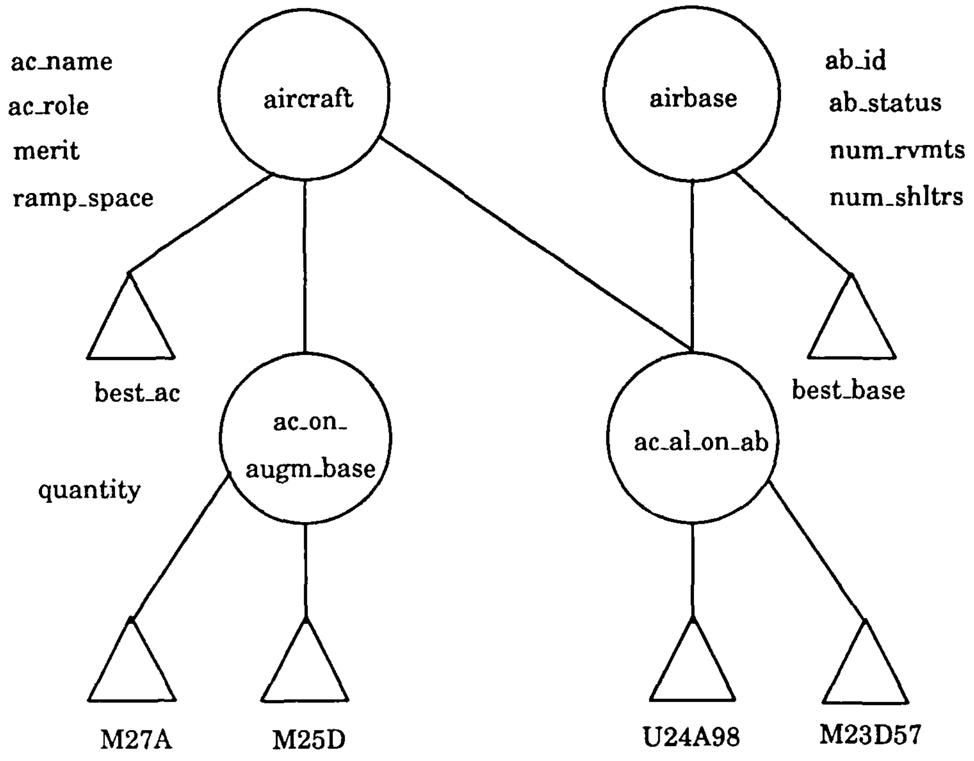


Figure 25. Classes for Automating Aircraft Beddown

were actually children of the two primary classes, airbase and aircraft, enabling them to efficiently inherit common properties (See Figure 25.)

The objects "best_base" and "best_ac" are used to store the airbase and aircraft with the highest merit. The objects in the class *ac_on_augm_base* are identified by the name and role of the aircraft. The objects in the class *ac_al_on_ab* are identified by the name and role of an aircraft and the airbase id on which the aircraft may be stationed.

5.2.3 Rule Generation. The actual relocation of aircraft from the augmentation base to their destinations required the following actions:

1. Load in data from the TWX database on the aircraft at the staging base.
2. Find the aircraft with the highest merit.

3. Find all airbases on which the aircraft may be stationed.

4. Find the airbase with the highest merit

5. Move as many aircraft to the airbase as allowed by available ramp space.

6. Get next best airbase as needed.

7. Get next best aircraft as needed.

Figure 26 illustrates the flow of control needed to resolve the above requirements.

We again used the hypothesis *data_loaded* to start the knowledge session. The successful loading of all data concerning aircraft name, aircraft role, quantity, and ramp space used into the class *ac_on_augm_base* resulted in *data_loaded* being true. The decision table for rule number one is shown in Figure 27.

If the knowledge base was able to retrieve the needed material for the TWX database then *looking_for_best_ac* is placed on the system's agenda and investigated. This rule examines the merit of all aircraft within the ac_on_augm_base class. Upon finding the aircraft with the highest merit it places the name, role, merit, quantity, and ramp space needed in the object "best_ac." This rule is recursively used until it it cannot find an aircraft with a merit higher than the one held by "best_ac." The rule for determining possible sites to relocate the aircraft pointed to by "best_ac" are then evaluated (See Figure 28.)

Once the aircraft with the highest merit is found then all possible relocation sites for that aircraft are retrieved from the TWX database along with the base status, number of shelters and revetments, and ramp space available at those bases. The hypothesis for this rule is *get_possible_sites*. The airbases are placed in the class *ac_al_on_ab*. If one or more bases are found then the rule evaluates to true and the rule responsible for finding the base with the highest merit is placed on the system agenda (See Figure 29.)

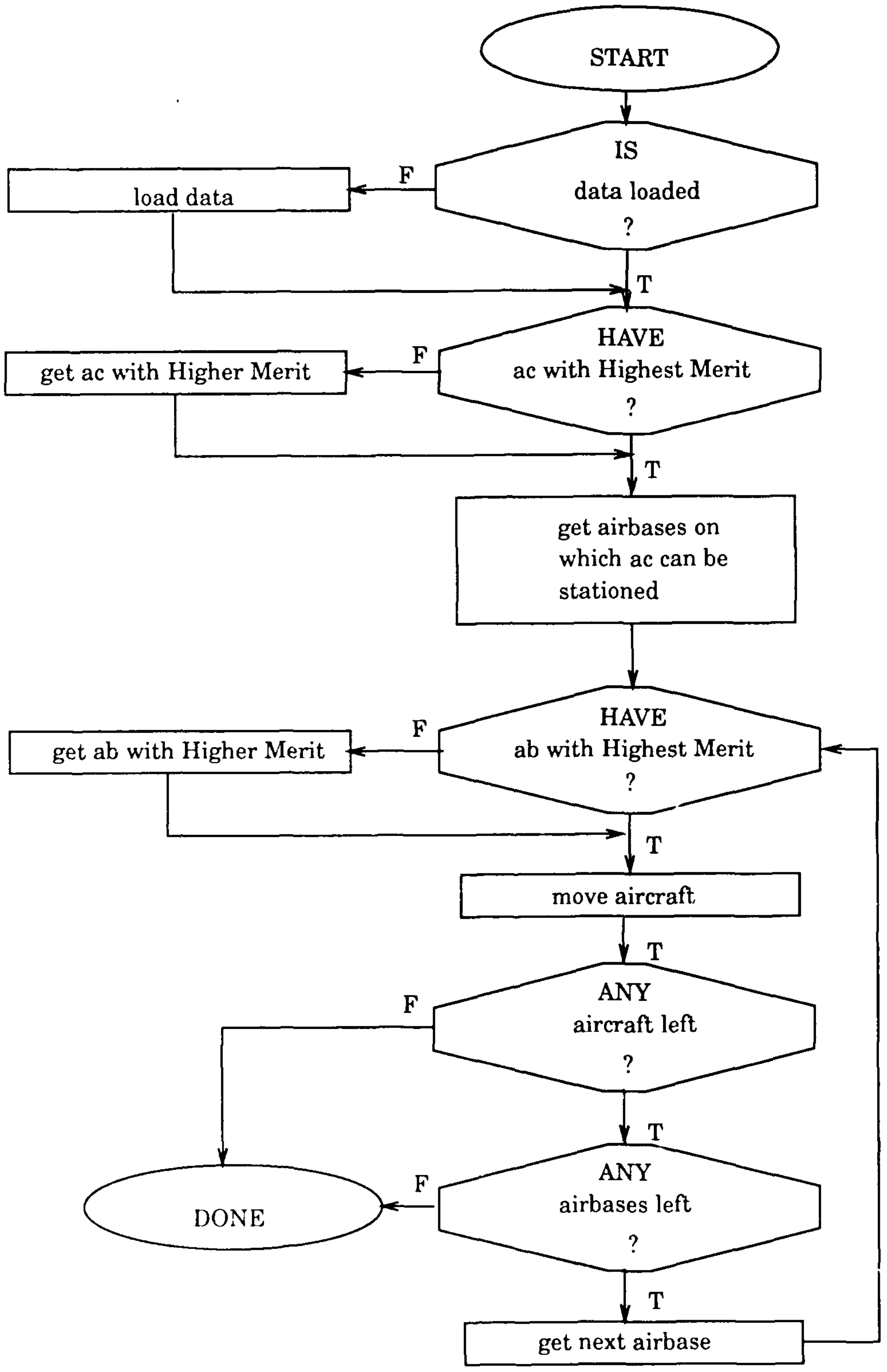


Figure 26. KB Rule Relationships for Automating Aircraft Beddown

CONDITIONS	HYPOTHESIS
READ in ac_name	data_loaded
READ in ac_role	
READ in quantity	RESET and FIRE rule to locate best ac
READ in rampspace needed	
READ in merit	

ACTIONS

Figure 27. Decision Table for *data_loaded*

CONDITIONS	HYPOTHESIS
Is ac.merit > best_ac.merit	looking_for_best_ac
Is ac.quantity > 0	
	best_ac.ac_name ⇐ ac.ac_name
	best_ac.ac_role ⇐ ac.ac_role
	best_ac.merit ⇐ ac.merit
	best_ac.rampspace ⇐ ac.rampspace
	best_ac.quantity ⇐ ac.quantity
	RESET and FIRE rule to get possible sites

ACTIONS

Figure 28. Decision Table for *looking_for_best_ac*

CONDITIONS	HYPOTHESIS
READ in ab_id	get_possible_sites
READ in ab_status	
READ in number of shelters	RESET and FIRE to locate best base
READ in number of revetments	
READ in rampspace available	

ACTIONS

Figure 29. Decision Table for *get_possible_sites*

CONDITIONS	HYPOTHESIS
Is ab.merit > best_base.merit	looking_for_best_base
Is ab.rampspace > 0	
	best_base.ab_id ⇐ ab.ab_id
	best_base.ab_status ⇐ ab.ab_status
	best_base.merit ⇐ ab.merit
	best_base.rampspace ⇐ ab.rampspace
	best_base.num_shltrs ⇐ ab.num_shltrs
	best_base.num_rvmts ⇐ ab.num_rvmts
	RESET and FIRE ac movement rule

ACTIONS

Figure 30. Decision Table for *looking_for_best_base*

The rule responsible for determining the airbase with the highest merit has the same structure as the rule for finding the aircraft with the highest merit. All bases within the *ac_al_on_ab* class are reviewed and the base with the highest merit is placed in the object "best_ab." The algorithm developed in the prioritization subsection above was used to formulate each airbase's merit. The rule is recursively fired until the airbase with the highest merit resides in "best_base." The hypothesis for this rule is *looking_for_best_base.* When this hypothesis evaluates to false the rules for relocating the aircraft are inspected. Figure 30 shows the decision table for this rule.

After the best airbase and aircraft have been established a rule is fired to calculate the maximum number of aircraft that can be sent to that base. *Move_planes_to_base* places this value in the object "max_num_of_ac." This rule then pursues two other rules to decide whether or not this amount can cover the total quantity of the aircraft at the augmentation base. Figure 31 shows the decision table for *move_planes_to_base.*

The next two rules have the following hypotheses: *move_all_ac_to_base* and *move_some_ac_to_base.* If the quantity of aircraft that needs to relocated exceeds the number that can be station on a particular airbase then only the number that the airbase can hold

CONDITIONS	HYPOTHESIS
IS best_ac KNOWN	move_planes_to_base
IS best_base KNOWN	
	CALCULATE max_num_of_ac
	RESET and FIRE plane movement rules

ACTIONS

Figure 31. Decision Table for *move_planes_to_base*

CONDITIONS	HYPOTHESIS
IS best_ac.quantity $\leq$ max_num_of_ac	move_all_ac_to_base
	best_ac.quantity $\Leftarrow$ 0
	best_base.rampspace $\Leftarrow$
	best_base.rampspace-rampspace used
	FIRE rule to get new ac

ACTIONS

Figure 32. Decision Table for *move_all_planes_to_base*

will be move from the augmentation base. However, if the number of aircraft that can be moved to a base is greater than the number awaiting relocation then all aircraft from the augmentation base will be transferred. Both rules update the following database values:

- the quantity of aircraft existing at the augmentation base
- the quantity of aircraft existing at the receiving base
- the ramp space available at the receiving base

The decision tables for these rules can be seen in Figures 32 and 33.

The final two rules in the knowledge base determine whether to retrieve another airbase for the current aircraft or acquire a new aircraft. If all planes of a specific name and role were removed then the hypothesis *check_for_new_ac_needed* would evaluate to true, effecting the deletion of the object "best_ac" and the class *ac_al_on_base*. This would allow the rules, used previously, to again determine a new aircraft with the best merit and the bases available for

CONDITIONS	HYPOTHESIS
IS best_ac.quantity > max_num_of_ac	move_some_ac_to_base
	best_ac.quantity ⇐ best_ac.quantity-max_num_of_ac
	best_base.rampspace ⇐ best_base.rampspace-rampspace used
	FIRE rule to get new base
	ACTIONS

Figure 33. Decision Table for *move_some_planes_to_base*

CONDITIONS	HYPOTHESIS
IS best_ac.quantity = 0	check_for_new_ac_needed
	DELETE best_ac and ac_al_on_ab
	RESET and FIRE rule to get new ac
	ACTIONS

Figure 34. Decision Table for *check_for_new_ac_needed*

its beddown operations. If the quantity of "best_ac" has not been depleted then the hypothesis *check_for_new_base_needed* is true. This rule removes the object "best_base" and fires the rule for determining a new base and also resets and fires those responsible for aircraft movement (See Figures 34 and 35.)

The above sequence of rules continue to execute until either no planes exist at the augmentation base or the maximum number of aircraft that can be transferred are moved.

CONDITIONS	HYPOTHESIS
IS best_base.rampspace < rampspace needed	check_for_new_base_needed
IS best_ac.quantity > 0	DELETE best_base
	RESET and FIRE rule to get new base
	ACTIONS

Figure 35. Decision Table for *check_for_new_base_needed*

5.3 Solution

All data required by the knowledge base is retrieved through Nexpert's database bridge from the TWX relational database. The following paragraphs present the SQL statements necessary for obtaining the desire information in the needed format.

The name, role, merit, ramp space needed, and quantity of aircraft stationed at the augmentation base are generated by joining the *rd_aircraft* and *rd_ac_on_ab* table within the TWX database. The SQL statements necessary are:

```
select distinct b.ac_name,b.ac_role,a.merit,b.quantity,
       a.ramp_space
from rd_aircraft a, rd_ac_on_ab b
where a.ac_name = b.ac_name
and   a.ac_role = b.ac_role
and   b.ab_id = 96
order by merit DESC, ac_name, ac_role
```

Airbase number, 96, is the augmentation base. The objects are ordered by merit in descending order. This creates a more efficient knowledge base since the first object read in will have the highest merit and the rule responsible for finding the aircraft with the highest merit will always choose the first object.

Information pertaining to the airbases on which an aircraft can be stationed is retrieved by joining *rd_ac_al_on_ab*, *rd_airbase*, *rd_ac_on_ab*. The current version of the knowledge base retrieves data for all aircraft at the augmentation base and then removes the unneeded bases. The following is the SQL statements used for this operation:

```
select distinct a.ac_name+a.ac_role+ASCII(a.ab_id),
       a.ac_name,a.ac_role,a.ab_id,b.ab_status,
       b.num_shelters,b.num_revet,b.ramp_space
from rd_ac_al_on_ab a, rd_airbase b, rd_ac_on_ab c
where c.ab_id = 96
and   a.ac_name = c.ac_name
and   a.ac_role = c.ac_role
and    a.ab_id = b.ab_id
order by ac_name,ac_role,ab_status DESC,num_shelters DESC,
         num_revets DESC, ramp_space DESC
```

The first line of the SQL statement creates an unique key for each object read in from the database. The key is determined by the aircraft name, role, and by the airbase id. Again all data is ordered in a way to make the knowledge base work more efficiently.

The rule control structure was created using Nexpert's context editor. The strategy used in this knowledge base was "propagate when true" (PWT). This means that rules in context with a current rule will not be placed on the knowledge session's agenda unless the current rules evaluates to only true. Thus for the rule used in finding an object with the highest merit, the rules are placed in context with themselves and with the rules needed to continue with the session. As long as the rules evaluate to true then they are still looking for an object that has a higher merit than the current one. When the rules evaluate to false then the other rules are allowed to execute. This strategy is accomplished through the use of an "inference category." If two rules are in context with another rule and the rule evaluates to true then the next rule to be used is determined by which rule has the highest inference category. In the case of the rule looking for the highest merit, recursion is produced by making the current rule's inference category higher than any of the other rules in context with it. Below are a list of the hypotheses and their contexts. The number in parentheses is the inference category.

```
data_loaded:
    looking_best_ac (3)
looking_for_best_ac:
    looking_for_best_ac (3)
    get_possible_sites (1)
get_possible_sites:
    looking_for_best_base (3)
looking_for_best_base:
    looking_for_best_base (3)
    move_planes_to_base (1)
move_planes_to_base:
    move_all_ac_to_base (1)
```

```
          move_some_ac_to_base (2)
      move_all_ac_to_base:
          check_for_new_ac_needed (1)
      move_some_ac_to_base:
          check_for_new_base_needed (1)
```

Updates to the TWX database from the knowledge base are accomplished through the use of the object "update." By using the IF-CHANGE mechanism in Nexpert, whenever the value of "update" is changed from false to true, aircraft quantities on the receiving base are updated. A log file containing the destination ab_id, ac_name, ac_role, and quantity of aircraft moved is also updated and a screen is displayed to the user describing the action taken.

The rules for finding the object with the highest merit must evaluate to true at least once. This is accomplished by initializing the merits of "best_base" and "best_ac" to -1. Thus any aircraft or airbase with a merit greater than or equal to zero with cause to the rules to be evaluated to true. The merit for each airbase is automatically calculated when asked for by the knowledge session. Nexpert's ORDER-OF-SOURCES mechanism applies the algorithm developed whenever the knowledge session detects a needed value that is unknown. This allows the merit of the airbase to dynamically change during a session due to the number of aircraft relocated to the base.

The FORMS-INPUT program in Nexpert again provided the necessary statements to load the knowledge base and start the session. It was also used to display the current status of the session to the user.

5.4 Summary

One of the primary reasons for selecting a knowledge-based system instead of a procedural language for automating the AAFCE portion of TWX was the system's ability to facilitate changes without a major modification in software. After creating the above knowledge base, a suggestion was made to make the aircraft relocate in regiment-size flights of

CONDITIONS	HYPOTHESIS
IS max_num_of_ac ≥ 25	move_regiment_to_base
IS best_ac.quantity ≥ 25	
	best_base.rampspace ⇐
	best_base.rampspace-rampspace used
	best_ac.quantity ⇐
	best_ac.quantity-25*X
	RESET and FIRE rule to get new base

ACTIONS

Figure 36. Decision Table for *move_regiment_to_base*

CONDITIONS	HYPOTHESIS
HAVE all bases been used for regimental moves	check_for_all_bases_used
	RESET and FIRE rules for regular movement

ACTIONS

Figure 37. Decision Table for *check_for_all_bases_used*

25 planes instead, moving as many planes to a base as possible. The only modifications to the knowledge base were the addition of two rules and the placement of these rules within the context of the established ones. The first new rule determines the number of flights that a base can handle and move those flights to the receiving base. The hypothesis for this rule is *move_regiment_to_base*. The second rule is evaluated to true whenever all bases within the class *ac_al_on_ab* have been examined for moving regiment-size flights to them. If aircraft still exist then the old aircraft movement rules are allowed to fire in order to move as many planes as possible to bases with the highest merit. The hypothesis for this rule is *check_for_all_bases_used*. The decision table for these rules can be seen in Figures 36 and 37.

The current knowledge base relies on the red player for selecting the merit of each aircraft at the augmentation base. A future modification might be to generate the rules necessary for calculating the merit of the aircraft according to a set of criteria much like the list used to produce the merit of each airbase. This would create a more complete and independent knowledge base.

VI. *Logistics Movement*

The final event in the AAFCE phase is the movement of logistics from supply bases. Each aircraft requires a specified amount of petroleum (POL), munitions, and spares (PMS) to generate one sortie. Since it is possible for an aircraft to fly more than one sortie per day then the maximum amount of PMS must be available on the base to maintain the aircraft's maximum number of sorties. Munitions for an aircraft are determined by the type of mission it flies. Thus a base must be able to supply munitions for the mission that requires the highest load. POL and spares remain the same for each mission.

Aircraft moved from the augmentation base are transferred to their new bases with only two days worth of spares. All other supplies must be provided by the airbase. However, any aircraft not moved from the augmentation base requires the use of existing base supplies, including spare parts. If a shortfall occurs, the amount of PMS needed must be brought in from the supply base. There are two supply bases available to the red player. The supply base used is determined by the ATAF in which an airbase belongs. If the airbase is assigned to 2ATAF then its supply base is PAF AD or base number 21. If the airbase is a member of the 4ATAF then its supply base is PAF GA or base number 98.

PMS on each base must not exceed the base's maximum tonnage limit. If the required amount of supplies surpasses this limit, unessential supplies must be returned to the supply base before the needed items can be received.

Red players using TWX called for a knowledge base that would automate the m [illegible] nt of logistics by:

- Finding the maximum mission loads required by all aircraft on each base
- Locating all base PMS shortfalls
- Transferring unessential material back to the supply base if necessary

• Moving the required amounts of PMS to each base

6.1 *Analysis*

6.1.1 Object-Oriented Design. Numerous tables within the TWX database were required for supplying the knowledge base with the necessary data for automating logistics movement. Data on red PMS came from four different tables in the database. Weight measurements on specific PMS items came from the table *rd_pms*. The airbase inventories of PMS came from the table *rd_pms_on_ab*. The different types of munitions loads based upon an aircraft type and its mission came from the table *rd_std_lds*. The amount of munitions required by a specific mission came from the table *rd_std_mun*.

Other details such as the maximum tonnage limit for the base came from the table *rd_airbase*. Finally the table *rd_ac_on_ab* was used to provide the data for determining the maximum PMS needed and the existing PMS tonnage for each base.

The knowledge base used three classes to store this data. The classes *airbase* and *pms* were created as parents for the class *pms_diff_on_ab*. *Rd_pms_diff_on_ab* contains the airbase id number and the difference between the actual amount of PMS and the needed amount of PMS for each type of munitions, pol_diff, spr_diff, and aimi_diff, etc. The class also contains the ataf number of the airbase and the existing and maximum tonnage at the base. The class *pms* contains the name of each PMS item and its surface transportation weight. This class is used in determining whether or not the base tonnage limit will be exceeded by the amount of PMS required to cover all base shortfalls. Figure 38 shows the class structures and their relationships to their objects in the knowledge base

6.1.2 Rule Generation. The movement of needed materials to meet base shortfalls as well as the movement of overages required the following actions:

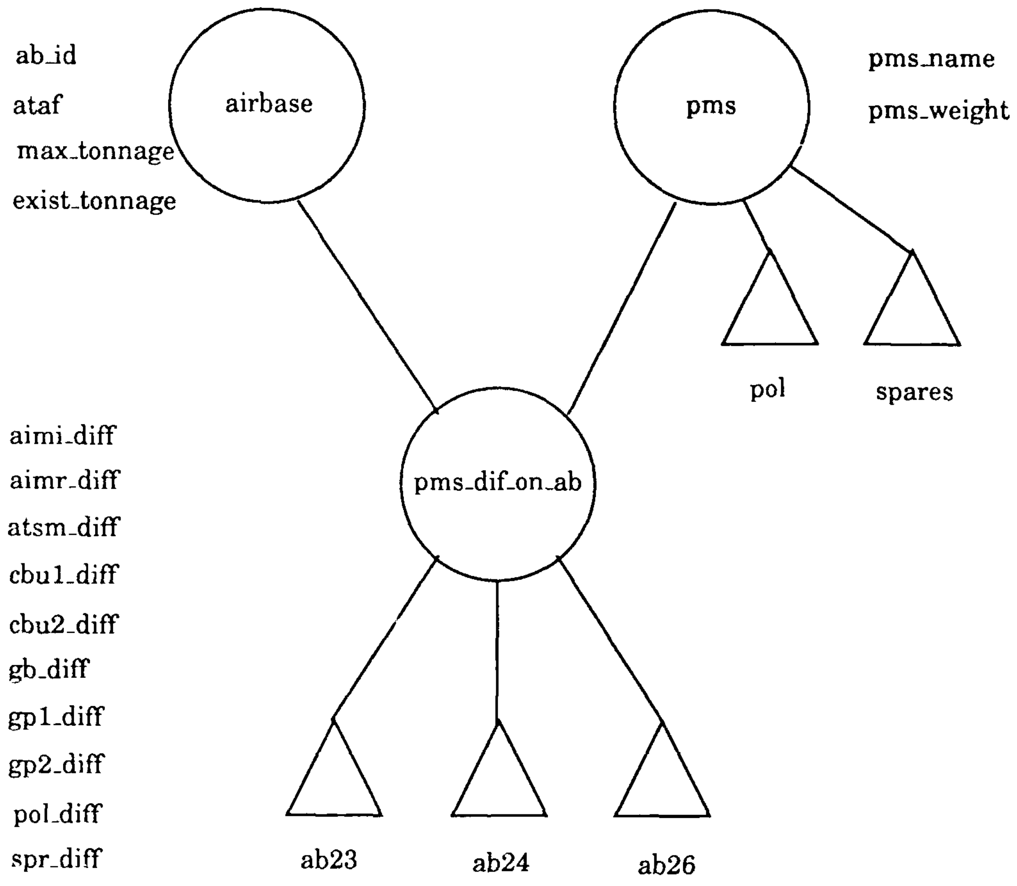


Figure 38. Classes for Automating Logistics Movement

1. Load in data concerning airbases and pms, eg. id's, weights, etc.
2. Load in PMS differences for each base
3. Check for PMS shortfalls
4. Move overages to supply base as needed
5. Satisfy base shortages
6. Get next airbase as needed

Figure 39 presents the flow of rule control used by the knowledge base for the above actions.

The knowledge session is started by suggesting the hypothesis *data_loaded*. This rule loads the classes *pms_diff_on_ab* and *pms* with data from the TWX database. A successful loading of the knowledge base classes results in the contextual propagation of the next rule (See Figure 40.)

Once the data has been loaded, the first base is set as the current base. The knowledge base then retrieves the PMS difference values for the current base. These values, calculated by the database, are the result of subtracting the needed supplies from the existing supplies. A positive difference denotes an overage while a negative number marks a shortage. These values are read for each base, resulting in a true evaluation of the hypothesis *current_base_set* (See Figure 41.)

The true evaluation of *current_base_set* results in the firing of eleven different rules. The first ten rules check for shortfalls in the ten types of PMS. All rules except for the rule that examines the POL supply may fire other rules selected for moving overages due to an excessive amounts of supplies that inhibit the movement of needed items. There is no need for POL overage movement rule since POL can be moved onto a base without increasing a base's supply tonnage. The final rule evaluates whether or not the current base has been absolved of all shortfalls. Once all shortfalls have been removed, another base is evaluated until no

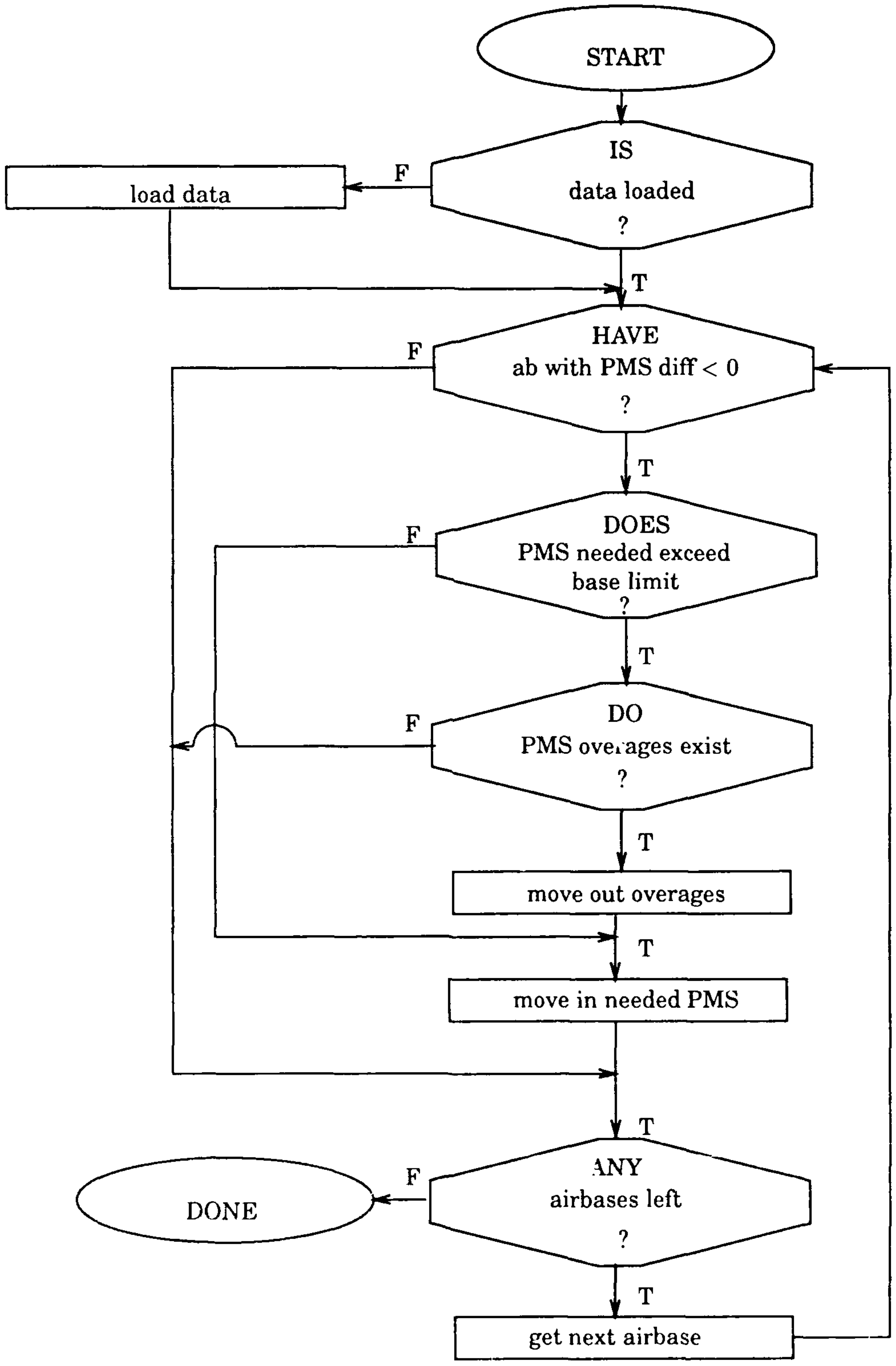


Figure 39. RB Rule Relationships for Automating Logistics Movement

CONDITIONS	HYPOTHESIS
READ in airbase info	data_loaded
READ in PMS info	
	FIRE rule to add in PMS differences

ACTIONS

Figure 40. Decision Table for *data_loaded*

CONDITIONS	HYPOTHESIS
IS airbase available	current_base_set
	LOAD PMS differences
	RESET and FIRE rules to find shortfalls

ACTIONS

Figure 41. Decision Table for *current_base_set*

other bases are found. The rules for POL movement, munitions movement with or without overages returned, and the next-base-selection rule are discussed in further detail below.

The hypothesis for POL shortfall evaluation is called *add_pol_from_supply_base*. If the POL difference is less than zero then this rule becomes true. The current airbase's id and quantity needed are then placed in a temporary object, dynamically named after the airbase id no, eg. POL23. This information is used by the rule responsible for supply base updates. Quantity needed is increased by ten percent which "pads" a base's supply. This pad was entered at the request of the red experts from the Air Force Wargaming Center, due to the random nature of attrition. Figure 42 shows the decision table for *add_pol_from_supply_base*. This rule then fires the rule responsible for updating the TWX database.

All other rules, accountable for munitions and spares, must first check to make sure the needed supplies do not surpass the airbase's tonnage limit. If the munitions' or spare's

CONDITIONS		HYPOTHESIS
IS current_base.pol_diff < 0		add_pol_from_supply_base
		CREATE OBJECT POLcurrent_base.ab_id
		ADD 10% to quantity needed
		RESET and FIRE rule to update supply base

ACTIONS

Figure 42. Decision Table for *add_POL_from_supply_base*

diff is less than zero and the maximum tonnage is exceeded by the needed supplies plus ten percent then the hypothesis *add_????_from_supply_base_with_over* becomes true. The variable, ????, is used in place of the actual munitions being evaluated. The valid set of munitions is AIMI, AIMR, ATSM, CBU1, CBU2, GB, GP1, GP2, and SPARES. If the munitions being evaluated was a cluster bomb unit, type 2, then the hypothesis would be *add_cbu2_from_supply_base_with_over*. The tonnage over the airbase maximum is placed in the object "tonnage_over_max" and the rule for selecting the largest overage at the current base is placed on the system's agenda. If there is enough room at the current base from the incoming supplies then the hypothesis *add_????_from_supply_base* is true and actions like those of *add_pol_from_supply_base* are executed. Figures 43 and 44 show the generic decision tables for the above rules.

The hypothesis *looking_for_largest_overage* evaluates to true until the largest difference munitions on the current airbase is found. The PMS name, PMS weight, and difference amount, are then assigned to the object "max_overage." The rule for sending the overages back to the correct supply base is then fired (See Figure 45.)

The hypothesis *overages_sent_back* evaluates to true when the object "max_overage" is defined. The rule is responsible for updating the current base's existing tonnage after the

CONDITIONS	HYPOTHESIS
IS current_base.pol_diff < 0	add__from_supply_base_with_over
IS max tonnage < tonnage needed	
	RESET and FIRE rule to find largest overage

ACTIONS

Figure 43. Decision Table for *add_????_from_supply_base_with_over*

CONDITIONS	HYPOTHESIS
IS current_base.pol_diff < 0	add_????_from_supply_base
IS existing tonnage > 0	
	CREATE OBJECT ????current_base.ab_id
	ADD 10% to quantity needed
	UPDATE existing tonnage
	RESET and FIRE rule to update supply base

ACTIONS

Figure 44. Decision Table for *add_????_from_supply_base*

CONDITIONS	HYPOTHESIS
Is current_diff > 0	looking_for_largest_overage
Is current_diff > max_overage	
	max_overage.diff ⇐ current_diff
	max_overage.name ⇐ pms_name
	max_overage.weight ⇐ pms_weight
	RESET and FIRE rule to send back overages

ACTIONS

Figure 45. Decision Table for *looking_for_largest_overage*

CONDITIONS	HYPOTHESIS
IS max_overage.name KNOWN	overages_sent_back
	UPDATE existing tonnage at current base
	RESET and FIRE rule to update supply base

ACTIONS

Figure 46. Decision Table for *overages_sent_back*

overages have been removed and firing the rules that update the supply base's inventory (See Figure 46.)

The rules responsible for updating the correct supply bases both point to the same hypothesis *supply_base_updated*. By having two rules point to the same hypothesis, an OR condition is created with two separate sets of actions. In this case, if the current airbase is part of the 2ATAF then supplies are sent to or retrieved from base 21. If the current base is a member of the 4ATAF then supplies are sent to and retrieved from base 98. The advantage of this OR condition allows both rules to be fired by propagating a single hypothesis. These rules update their respective supply bases as well as the PMS totals for the gaining/losing airbase. Once the database updates have taken place the knowledge session resets and fires *current_base_set*. This allows the updated differences to be re-loaded from the TWX database. The generic decision table for these rules is shown in Figure 47.

The last rule needed by the knowledge base designates the next airbase to be evaluated once the current bases is found to have no differences. The current base is then deleted from the class *pms_diff_on_ab* and the new airbase is selected. The hypothesis for this rule is *ready_for_next_base* (See Figure 48.)

CONDITIONS	HYPOTHESIS
IS current_base.ataf = 2 OR 4	supply_base_updated
	UPDATE supply base 21 OR 98
	UPDATE supplies of current base
	RESET and FIRE rule to check current base

ACTIONS

Figure 47. Decision Table for *supply_base_updated*

CONDITIONS	HYPOTHESIS
Are ALL shortfalls removed	ready_for_next_base
	DELETE current base
	RESET and FIRE rule to set next current base

ACTIONS

Figure 48. Decision Table for *ready_for_next_base*

6.2 Solution

Airbase information for the class *pms_diff_on_ab* was generated by joining the tables *rd_airbase*, *rd_pms*, and *rd_pms_on_ab*. Properties such as airbase id, ATAF number, and maximum tonnage came from the table attributes of *rd_airbase*. Existing tonnage was calculated by multiplying PMS surface weights from rd_pms with the airbase inventory of PMS found in *rd_pms_on_ab*. The SQL statement used to generate this information is:

```
select a.ab_id, max(a.ataf), max(a.max_tonnage),
       sum(b.quantity*c.sur_weight)
from rd_airbase a, rd_pms_on_ab b, rd_pms c
where a.ab_type = 1
and   a.ab_id = b.ab_id
and   b.pms_name = c.pms_name
group by a.ab_id
```

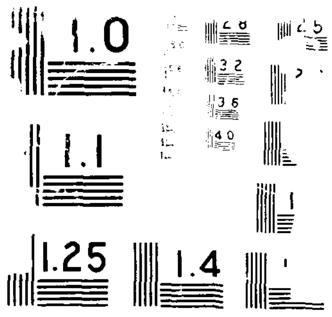
The function *max* is used since the *group by* clause requires all properties within the *select* clause to be either defined as one of its arguments or a function. Thus, *max* is used to satisfy this requirement even though it never changes the value of the properties, ie. the maximum ATAF number of those bases within the 2 ATAF is 2.

The class *pms* retrieved its data from the table *rd_pms* using the following SQL statement:

```
select distinct pms_name, pms_weight
from rd_pms
where pms_name != "POL"
and   pms_name != "OTHER"
and   pms_name != "RX"
and   pms_name != "STDA"
order by pms_name
```

POL's weight is defined as zero when shipment is by surface vessels. Thus, it cost nothing to transport. RX, STDA, and other are never used in the current version of TWX.

AD-A215 728 AN EXPERT SYSTEM FOR AUTOMATING NUCLEAR STRIKE AIRCRAFT 2/2
REPLACEMENT AIRCR..(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. H D HARKEN
UNCLASSIFIED DEC 89 AFIT/GCS/ENG/89D-7 F/G 1/2 NL


1.0
2.8
2.5
3.2
3.6
4.0
1.1
1.25
1.4

so are not needed by the knowledge base. However, removing these four lines will allow the supplies to be used whenever the need arises.

The SQL statements needed to retrieve POL and spare differences at each supply base required data from the tables *rd_airbase*, *rd_ac_on_ab*, *rd_aircraft*, and *rd_pms_on_ab*. The amount of POL needed by an aircraft is based on the number sorties that can be flown by that aircraft and a surge factor. The number of spares required by a aircraft is also dependent upon the number of sorties flown. The next two SQL statement show how data was retrieved from the TWX database for POL and spares respectively.

```
select a.ab_id,
       max(d.quantity) -
       sum(b.quantity*c.sortie_rate*c.surge_factor*c.pol_sor)
from   rd_airbase a, rd_ac_on_ac b, rd_aircraft c,
       rd_pms_on_ab d
where  a.ab_type = 1
and    a.ab_id = b.ab_id
and    b.ac_name = c.ac_name
and    b.ac_role = c.ac_role
and    d.ab_id = a.ab_id
and    d.pms_name = "POL"
group  a.ab_id

select a.ab_id,
       max(d.quantity) -
     sum(b.quantity*c.sortie_rate*c.surge_factor*c.spares_sor)
from   rd_airbase a, rd_ac_on_ac b, rd_aircraft c,
       rd_pms_on_ab d
where  a.ab_type = 1
and    a.ab_id = b.ab_id
and    b.ac_name = c.ac_name
and    b.ac_role = c.ac_role
and    d.ab_id = a.ab_id
and    d.pms_name = "SPARES"
group  a.ab_id
```

PMS differences between existing and needed supplies were created by joining the database tables *rd_airbase*, *rd_ac_on_ab*, *rd_std_lds*, and *rd_std_mun*. The properties needed for the separate munitions were stored in a database structure called a *view*. This structure is used to define a "virtual table" within the database that can be merged with other SQL statements to produce data which cannot be created with a single SQL statement. The SQL code used to create the view was:

```
create view max_mun (ab_id, ac_name, ac_role, mun_name,
                     quantity)
as select a.ab_id, a.ac_name, a.ac_role, d.mun_name,
          max(d.quantity)*max(a.quantity)
from rd_ac_on_ab a, rd_airbase b, rd_std_lds c,
     rd_std_mun d
where a.ab_id = b.ab_id
and   b.ab_type = 1
and   a.ac_name = c.ac_name
and   a.ac_role = c.ac_role
and   c.pref_load = d.load_num
group by a.ab_id, a.ac_name, a.ac_role, d.mun_name
```

This view called *max_mun* stores the amount of munitions needed for each plane on a base. This amount is the maximum number obtained by checking the standard loads for every mission an aircraft might be called upon to fly. The actual data used by the knowledge base is determined by subtracting the amount in the *max_mun* view from the actual inventories found in the table *rd_pms_on_ab*. The knowledge base must make eight separate queries in order to retrieve the desired information on all eight munitions. The following SQL statement shows a query for airbase shortfalls/overages of general purpose bombs, type 1:

```
select m.ab_id, max(m.mun_name),
       max(a.quantity) - sum(m.quantity)
from max_mun m, rd_pms_on_ab a
where m.mun_name = "GP1"
and a.pms_name = m.mun_name
and a.ab_id = m.ab_id
group by m.ab_id
```

The rule control strategy again was set to "propagate when true", as described in the last chapter. All contexts of a current rule were placed on the system's agenda only if the rule evaluated to true. Inference categories were use to make sure the rules responsible for checking overages fired before those responsible for logistic movement to the base. Below are a list of hypotheses. Their inference categories are shown in parentheses if their are the categories are greater the one (the default).

```
add_pol_from_supply_base:
     supply_base_updated
add_spares_from_supply_base:
     supply_base_updated
add_spares_from_supply_base_with_over:
     looking_for_largest_overage
add_????_from_supply_base:
     supply_base_updated
add_????_from_supply_base_with_over:
     looking_for_largest_overage
current_base_set:
     add_pol_from_supply_base
     add_spares_from_supply_base
     add_spares_from_supply_base_with_over(2)
     add_????_from_supply_base
     add_????_from_supply_base_with_over(2)
     ready_for_next_base(-1)
data_loaded:
     current_base_set
looking_for_largest_overage:
     looking_for_largest_overage(3)
     overages_sent_back
overages_sent_back:
      supply_base_updated
ready_for_next_base:
      current_base_set
supply_base_updated:
      current_base_set
```

Updates to the TWX database again used the IF-CHANGE utility and the knowledge base object "update". After movement calculations were performed, the object's boolean value was changed by a rule to true. This action initiated updates to the tables responsible for PMS differences and the current base's tonnage limit.

The rule responsible for finding the largest overage available must be evaluated to true at least once in order to propagate the next rule after the largest overage is found. This again required the initialization of object properties using the ORDER-OF-SOURCES mechanism. With this program "max_overage.diff" was set to zero. This ensured a true evaluation of the rule if there existed at least one positive difference, eg. an overage.

The initiation of the knowledge session was implemented using the FORMS-INPUT program in Nexpert. After loading the knowledge base, the program suggested *data_loaded*, which launched the knowledge process.

6.3 Summary

The current version of the logistics knowledge base relies on the TWX constraint that all shortfalls can be alleviated. This, however, is not always the case in the real world. A suggested enhancement might be to allow PMS transfers from bases other than the supply base that have overages or might not be capable of generating aircraft sorties due to substantial damage. This would improve the knowledge base's emulation of actual battle scenarios and release it from the TWX environment.

VII. Conclusions and Recommendations

7.1 Summary

The main focus of this research was providing a means of automating the AAFCE phase of the Theater Warfare Exercise. By utilizing an AI expert system shell, the goals as stated in the introduction chapter of this thesis were realized. Three knowledge bases were created through the use of the Nexpert Object development environment. Each knowledge base, independent of the others, fulfilled the requirement as set forth by the personnel at the Air Force Wargaming Center.

The actual automation of the planning section of the Theater Warfare Exercise could have been realized through the use of a procedural language and not a rule based expert system. However, this research was cited as the first of many with the final goal of totally automating the red side of TWX. The automation of target and aircraft selection, along with actual strategy evaluation would have been severly restricted if the only developmental tool used was a simple programming language. This research's largest contribution was finding a flexible developmental platform for the work ahead and creating a design model and its methodologies that will facilitate the development process for those who follow.

The knowledge base for the automation of nuclear strike aircraft replacement maintains a very simple, but effective, heuristic for sustaining the desired number of aircraft at their designated bases. The addition of this knowledge base will provide the red player with an extra fifteen to thirty minutes of time that can be devoted to the ATAF phase of TWX.

The knowledge base responsible for aircraft beddown increases the amount of extra time that can be utilized by the red player by a minimum of one hour. This represents a decrease in AAFCE planning of over twenty percent.

The logistics movement knowledge base removes the most mechanical section of the AAFCE phase. The red player, now, does not have to waste thirty minutes to an hour on a section that requires no more strategy that the ability to add and subtract, but is none the less a time consuming facet of the AAFCE phase.

The ability to execute three programs that complete their necessary functions within minutes after they have been initiated is a great improvement over the two to three hours of work spent looking at numerous reports, worksheets, and computer displays. The freedom from these tedious tasks will permit the red players to provide a higher quality exercise since a significant amount of their time will now be spent on target selection and prioritization.

7.2 Recommendations for Further Work

This thesis only completes the first step in automating the red player for TWX. Now that a developmental environment has been evaluated and a functioning application has been produced, the ATAF section of TWX should be automated. The selection of aircraft for a given mission and later the selection of the actual mission should be considered as the next two levels of automation within TWX

Nexpert Object provides numerous means by which an expert system shell can be executed. With the transition of the TWX database from the Ingres RDBMS to the Oracle RDBMS, the opportunity for creating a new platform for developing, generating, and operating new expert shells is available. Using the relational database's Applications-By-Forms tools and embedded code might provide a standardized means of utilizing an expert shell within TWX.

Finally, the area of exercise evaluation and comparison should be addressed with the use of expert system shells. Using a fully automated version of TWX with the same red strategies should allow red players to evaluate games played by different student seminars, thus providing a means to determine which student team was the best. A second application of this type of system might be to automate the blue side of TWX and judge the merit of different red strategies. This would render a more effective and flexible lesson to the blue teams.

Appendix A. *User's Manual*

A.1 Introduction

The software for this thesis was originally slated for use on a DEC GPX workstation networked to a Microvax III which hosted the Ingres RDBMS version of TWX. However, the personnel at the Air Force Wargamming Center received some new equipment; specifically Sun 386i workstations. The decision to transfer TWX to the Sun's became a little more difficult when it was determined that the workstations would use the Oracle RDBMS instead of Ingres. This required a complete makeover of the TWX database and a revision in the Nexpert Object software order. Nexpert was originally to be hosted on the GPX workstation under the VMS operating system, but the Sun's are a Unix machine. Luckily, one of the many facets of Nexpert was that it runs on several machines, and the Suns happened to be one of those platforms. HOWEVER, the Oracle and Nexpert software never arrived before this thesis was completed.

Thus, this thesis was based on the IBM AT version of Nexpert with all database communications simulated using Nexpert's database base format (see the programmer's manual for more information.) The SQL statements developed by this thesis were fed to the TWX database on the DEC Microvax III. The results were stored in files and downloaded to a Zenith 386 PC. The database files were then formatted into Nexpert's database structure and use by the knowledge bases.

The rest of this manual is broken into two sections. The first section deals with where to find the SQL statements used to create the data files needed by the knowledge bases and how they were created. The second section deals with how Nexpert was implemented on the microcomputer and where to locate the numerous files needed to run the knowledge bases.

It should be noted that while the three knowledge bases created by this thesis are only simulations, the rules used to generate the sessions are correct and will only need a small amount of changing when they are uploaded to the Sun 386i. The only reason that these sessions are considered simulations is that the data used by the knowledge bases is not dynamically retrieved from and saved in the actual TWX database.

A.2 *TWX Database Files and Operations*

All data files used by Nexpert were generated using the Ingres Interactive SQL program (ISQL) on a DEC Microvax III. The Microvax which hosts the TWX database is called RAVEN. A seminar was created from the TWX master database, TWXMSTR. The seminar number used for this thesis was 3. A seminar is created by using the TWX database control menus. The TWX controller is executed by entering the following command:

twxcom

Entering option 1, "Create a new seminar database", produces a prompt asking for the seminar number. This command creates the seminar by creating the database, *TWX3*, where the seminar number chosen was 3. All SQL statements are then applied to *TWX3*. When using the ISQL program it is best to first change directories to a place where you can save and retrieve session outputs to files. All data files were saved in a directory on RAVEN. The command for setting the default directory to the TWX source directory is:

set def *DUA1:[mroth.dharken.twx]*

The directory is shown below:

```
Directory DUA1:[MROTH.DHARKEN.TWX]

AATTACK.SQL;1       AG.DB;1             ASTRIKE.SQL;2       AUG.SQL;1
DIFAIMI.DAT;1       DIFAIMI.SQL;1       DIFAIMR.DAT;1       DIFAIMR.SQL;1
```

```
DIFATSM.DAT;1       DIFATSM.SQL;1       DIFCBU1.DAT;1       DIFCBU1.SQL;1
DIFCBU2.DAT;1       DIFCBU2.SQL;1       DIFGB.DAT;1         DIFGB.SQL;1
DIFGP1.DAT;1        DIFGP1.SQL;1        DIFGP2.DAT;1        DIFGP2.SQL;1
DIFMUN.SQL;1        DIFPOL.SQL;3        DIFSPR.SQL;1        MAX_TON.SQL;4
MUNVIEW.SQL;1       RAMP_SP.SQL;3       RD_AC_AL_AB.DAT;2   RD_AC_AL_AB.SQL;5
RD_AIRBASE.DAT;2    RD_AIRBASE.SQL;2    RD_AIRCRAFT.DAT;2   RD_AIRCRAFT.SQL;2
RD_PMS.SQL;1        README.LIS;3        RSTRIKE.SQL;1       TEMP.AB;2
TEMP.MUN;1          TEMP.PMS;1          TEMP.POL;2          TEMP.SPR;1

Total of 41 files.
```

A listing of what each file contains can be found by looking at the file README.LIS. This can be accomplished by typing:

type /pa *readme.lis*

The ISQL environment is executed by entering:

isql *twx3*

You can then create, load, display, or output any legal SQL statement that uses data from the *TWX3* database. The outputs can be saved to a file which can then be transferred to the microcomputer using the Xmodem communications protocol. The initial command for downloading a file is:

xmodem *filename*

A secondary prompt then asks for the commands to send the file to the microcomputer. This command is simply:

st *filename*

It is useful to think of the command **st** as "sent text." Once data has been downloaded to the microcomputer, it is formatted for use by Nexpert Object. This is discussed in the programmer's manual.

A.3 *Nexpert Files and Operations on the PC*

The microcomputer environment consists of the following:

- One Zenith 386 PC with 1 Meg of memory on the mother board, one 360K floppy disk drive, and one 1.2M floppy disk drive
- 3 Megabytes of Expanded Memory
- Zenith MS-DOS Version 3.30+
- Microsoft Windows 386 Version 2.0
- Logitech Mouse
- 80 Megabytes of hardisk space
- DEC LN03R postscript printer

Nexpert Object runs under the windowing environment provided by Microsoft Windows. The only changes necessary to Microsoft Windows is to add the following lines to the file, win.ini, in the windows directory:

```
kb=e:\nexpert\nexpert *.kb
frm=e:\nexpert\nexpert *.frm
```

These commands, placed in the *extension* section, tell windows to execute nexpert whenever files with the extensions *.kb and *.frm are selected. The filename of the selected file will then be passed to Nexpert as a parameter, thus loading the knowledge base or input form.

All Nexpert files can be found in the directory, *e:\nexpert*. It is important the your DOS environmental variable, *PATH*, contain the Nexpert and Windows Directories. Otherwise you will see numerous FILE NOT FOUND errors.

For each major section of this thesis, a unique knowledge base was created. These knowledge bases can be found in their own directories on drive E. The following shows the directory pathname and a brief description of the three knowledge bases:

- **e:\hharken\nxpfiles\strike** - knowledge base for the automation of nuclear strike aircraft replacement
- **e:\hharken\nxpfiles\beddown** - knowledge base for the automation of aircraft bed-down
- **e:\hharken\nxpfiles\log** - knowledge base for the automation of logistics movement

The directory containing the strike aircraft replacement knowledge base is shown below:

```
 Volume in drive E is AFIT_ENG
 Directory of  E:\HHARKEN\NXPFILES\STRIKE

.            <DIR>        8-29-89  10:38a
..           <DIR>        8-29-89  10:38a
DEMO     BAT        16    8-28-89  12:23p
RESET    BAT       170    8-25-89   3:49p
CONVERT  EXE      9287    8-24-89   2:48p
AATTACK  BAK       631    8-24-89   5:09p
ASTRIKE  BAK       631    8-28-89  12:39p
AUG      BAK       477    8-28-89  12:20p
NXPDB    BAK       103    8-23-89   6:24p
RSTRIKE  BAK       631    8-04-89   4:19p
STRIKE   BAK     10968    8-28-89   5:36p
CONVERT  C        1244    8-24-89   2:47p
AATTACK  DB        631    8-24-89   5:09p
ASTRIKE  DB        631    8-28-89  12:39p
AUG      DB        477    8-28-89  12:20p
AUGM     DB        103    8-23-89   6:24p
REROLE   DB        103    8-23-89   6:24p
RSTRIKE  DB        631    8-04-89   4:19p
AUGM     FRM      1503    8-28-89   5:44p
```

```
RBROLE    FRM      1399   8-28-89   5:44p
STRIKE    FRM      1196  10-06-89   2:41p
STRIKE    KB      10920   8-28-89   5:36p
          22 File(s)    120832 bytes free
```

DEMO.BAT is an executable batch file that runs windows and loads nexpert with the input form that will start the knowledge session.

RESET.BAT is an executable batch file that resets the data files (**.DB*) by copying the **.BAK* files to their respective filenames, ie. *AATTACK.BAK* ⇒ *AATTACK.DB*. Before runnng the demo, *RESET.BAT* should be executed first.

CONVERT.EXE is an executable 'C' program that takes reports generated by Nexpert and wraps the output lines to 80 characters so they can be printed on a dot-matrix printer. The source code for this program is in *CONVERT.C*.

The **.DB* files are the data files used by Nexpert Object. Theses files are in the Nexpert Database format. The data files are read and updated during a knowledge session. The **.BAK* files are used to reset the data after a knowledge session has run. *ASTRIKE.DP* contains data on the actual number of strike aircraft at an airbase. *AATTACK.DB* contains data on the actual number of attack aircraft at an airbase. *AUG.DB* contains the type and number of attack aircraft available at the augmentation base. *AUGM.DB* and *REROLE.DB* are log files that keep track of the type of aircraft and quantity of aircraft moved for the augmentation base and reroled respectively. *NXPDB.BAK* is used to create these files before a session. *RSTRIKE.DB* is the number of strike aircraft required at an airbase.

The **.FRM* files are the input forms used by Nexpert. These files are command scripts that Nexpert compiles that can load and execute a knowledge base. These files are also responsible for the display of session information to the user. *STRIKF.FRM* is the file responsible for the loading and the execution of the knowledge base. *REROLE.FRM* displays the type and quantity of aircraft that are reroled on airbase. *AUGM.FRM* displays the receiving airbase number, aircraft type, and quantity of aircraft moved from the augmentation base.

STRIKE.KB is the ASCII file containing the knowledge base used by Nexpert Object. This file can be ported to other hardware platforms running Nexpert and then successfully loaeded on the new machine.

The directory containing the aircraft beddown knowledge base is shown below:

```
 Volume in drive E is AFIT_ENG
 Directory of  E:\HHARKEN\NXPFILES\BEDDOWN

.            <DIR>       8-29-89  10:39a
..           <DIR>       8-29-89  10:39a
DEMO1    BAT        28   9-14-89   2:23p
DEMO2    BAT        28   9-14-89   2:23p
RESET    BAT       117   9-11-89   3:39p
CONVERT  EXE      9287   8-24-89   2:48p
ACALLOW  BAK      7388   9-12-89   5:15p
AUGMBASE BAK       672   9-07-89   5:25p
NXPDB    BAK       175   9-07-89   4:44p
ACALLOW  DB       7388   9-12-89   5:15p
AUGMBASE DB        672   9-07-89   5:25p
AC_ON_AB DB        175   9-07-89   4:44p
BEDDOWN  FRM      1551   9-06-89   4:53p
STARTUP1 FRM      1241  10-06-89   2:39p
STARTUP2 FRM      1241  10-06-89   2:40p
BEDDOWN1 KB      11688   9-14-89   2:22p
BEDDOWN2 KB      14722   9-14-89   2:03p
       17 File(s)     120832 bytes free
```

DEMO1.BAT executes the demo for the knowledge base, BEDDOWN1. This knowledge base beds down aircraft as quickly as possible. *DEMO2.BAT* executes the demo for the knowledge base, BEDDOWN2. This knowledge base beds down aircraft according to Red regiment size requirements. *DEMO1.BAT* and *DEMO2.BAT* executes *STARTUP1.FRM* and *STARTUP2.FRM* respectively.

ACALLOW.DB contains data on aircraft types and the airbases where they are allowed to be stationed. *AUGMBASE.DB* contain the type and quantity of aircraft at the augmentation base that need to be moved. *AC_ON_AB.DB* is the log file that keeps track of aircraft movement by recording the receiving base number, aircraft type and aircraft quantity.

STARTUP1.FRM loads and executes *BEDDOWN1.KB* and *STARTUP2.FRM* loads and executes *BEDDOWN2.KB*. *BEDDOWN.FRM* is responsible for displaying the receiving air-base number, type of aircraft, and quantity of aircraft sent from the augmentation base.

The directory containing the logistics movement knowledge base is shown below:

```
 Volume in drive E is AFIT_ENG
 Directory of  E:\HHARKEN\NXPFILES\LOG

.            <DIR>        9-15-89   3:56p
..           <DIR>        9-15-89   3:56p
DEMO     BAT        27    9-25-89   2:15p
RESET    BAT       483    9-20-89   8:04p
DIFATSM  BAK        73    9-20-89   2:30p
DIFAIMI  BAK       632    9-22-89   1:10p
DIFAIMR  BAK       613    9-22-89   1:23p
DIFCBU1  BAK       433    9-20-89   2:32p
DIFCBU2  BAK       433    9-20-89   2:35p
DIFGB    BAK       325    9-20-89   2:37p
DIFGP1   BAK       433    9-20-89   2:39p
DIFGP2   BAK       325    9-20-89   2:41p
DIFPOL   BAK       141    9-20-89   8:12p
DIFSPR   BAK       169    9-22-89   1:09p
TONNAGE  BAK       365    9-22-89   1:11p
WEIGHTS  BAK       325    9-20-89   6:52p
NXPDB    BAK       157    9-20-89   4:33p
BASE21   DB        468    9-22-89   4:43p
BASE98   DB        312    9-22-89   4:42p
DIFAIMI  DB        632    9-22-89   4:42p
DIFAIMR  DB        613    9-22-89   4:43p
DIFATSM  DB         73    9-20-89   2:30p
DIFCBU1  DB        433    9-20-89   2:32p
DIFCBU2  DB        433    9-20-89   2:35p
DIFGB    DB        325    9-20-89   2:37p
DIFGP1   DB        433    9-20-89   2:39p
DIFGP2   DB        325    9-20-89   2:41p
DIFPOL   DB        141    9-22-89   4:42p
DIFSPR   DB        169    9-22-89   4:43p
TONNAGE  DB        365    9-22-89   4:43p
WEIGHTS  DB        325    9-20-89   6:52p
STARTUP  FRM      1241   10-06-89   2:40p
LOG      KB      21319    9-25-89   2:28p
        33 File(s)    122880 bytes free
```

DEMO.BAT is an executable batch file that has Nexpert compile *STARTUP.FRM* which in turn loads and executes *LOG.KB*.

DIFF????.DB are the data files that contain the difference between the existing quantity of PMS and the required quantity, eg. *DIFFCBU1.DB* contains the differences for cluster bomb units, type 1, for all airbases. A positive number depicts an overage while a negative number shows a shortfall. *BASE21.DB* and *BASE98.DB* are log files listing the quantity of supplies being moved from and returned to their respective bases. A negative quantity shows supplies have been move to other bases. A positive quantity shows supplies have returned from other bases. *TONNAGE.DB* contains the existing tonnage and maximum tonnage for each airbase. *WEIGHTS.DB* contains the PMS name and weight for all legal supplies.

A.4 Summary

To execute a knowledge session, simply change to the directory containing the knowledge base desired, execute the command **RESET**, and enter the command, **DEMO**. This will hopefully reward you with a successful run. If not it might be wise to make sure all the files listed above are found in the correct directories.

Appendix B. *Programmer's Manual*

B.1 Introduction

This manual is not what one might expect after reading through those created for procedural languages. Most programmer's manuals are basically an application's code that has highly visible and readable comments. Unfortunately that is not the way the Neuron Data Corporation envisioned it. All components of a Nexpert knowledge base are encapsulated within a single ASCII file. However, this file was not created for the average programmer's reading pleasure. There are no facilities for the use of comments or indentation for legibility. In other words, there are only two things that can use this file; the Nexpert System software and a Neuron Data Corporation engineer.

Not all is lost; Neuron did provide a meager attempt at resolving this oversight by allowing the user to print a Nexpert editor's contents. These editors include:

- the Class Editor
- the Object Editor
- the Rule Editor
- the Context Editor
- the Property Editor

In the IBM AT version that I used, there was no way to send the output to a file. Thus I had to use a re-direction utility to send data destined for LPT1 into a MS-DOS file. Here I met with another problem. This time it was with Microsoft Windows. Output to the printer was sent in 256 character lines. You can imagine what this looked liked when printed on an 80 column CRT. To correct this I wrote a small C program that breaks lines every 79 characters and adds a carriage return/linefeed. This worked quite nicely until you tried to understand the contents of the files. I finally had to go and manually place line breaks and tabs within the files to make them legible.

I believe that many of these problems arose from the early version of Nexpert that I used. I am quite sure that most of aggravation I had will not be found once the Sun 386i version of Nexpert is installed. However, you as a programmer will still need to take the data files and manually indent and comment them so that another programmer can read and hopefully understand what you have accomplished.

The next five sections deal with each editor in Nexpert and how I used them to document my knowledge bases. The final section deals with the Forms Input Utility for controlling knowledge base execution and output. In all these sections I use my knowledge base responsible for nuclear strike aircraft replacement as an example.

B.2 The Class Editor

The class editor is the first utility used in creating a knowledge base. Here you create the classes and subclasses needed for the transferring data between the database and the knowledge base. The steps necessary for creating a class are as follows:

1. Start the class editor
2. Select the new class option from menu
3. Enter the class's name
4. Enter subclasses (if any)
5. Enter properties
6. Select the save class option from menu

After the selecting the *save class* option, you will prompted for the actual type of each property. The four types of properties used are numerical, string, boolean, and special. If you make a mistake when entering property types you will have to delete the property from the class and change the property type by calling up the property editor. If you name any subclasses they will be automatically created with properties from the parent class. This inheritance strategy can be changed, but I found no reason to do so.

By selecting the *print* option within the class editor and by redirecting the printer output to a file, I was able to document the classes within the knowledge base. There is an option to print the data to a file, but in the version I used, that particular function was not implemented. After saving the file I then placed comments within the code using the syntax for programs written in C. It should be noted that this file can never be used by the knowledge base and if you need to make a change, the above procedures will have to be repeated.

Below is the file that I created for the nuclear strike aircraft repl~~ement knowledge base's classes.

```
/***********************************************************************
        Name:  Classes for the Nuclear Strike Aircraft Replacement KB
      Author:  Capt H. Dallas Harken
        Date:  1 October 1989
     Version:  1.0

    Software:  Nexpert (IBM AT) Version 1.0

 Description:  This file contains all knowledge base classes for the
               nuclear strike aircraft replacement KB.  Properties
               and class relationships are also included.

               This file was created using the PRINT option within
               the Class Editor.
***********************************************************************/

CLASSES:

airbase
  PROPERTIES:
     ab_id = (Numerical)              /* Airbase Id Number */
  CHILDREN:
     stk_ac_on_ab
     atk_ac_on_ab
```

```
aircraft
  PROPERTIES:
     ac_name = (String)                /* Aircraft Name, Eg. M21 */
     ac_role = (String)               /* Aircraft Role, Eg. A,C,S,etc */
  CHILDREN:
     stk_ac_on_ab
     atk_ac_on_ab

atk_ac_on_ab
  PROPERTIES:
     ab_id = (Numerical)               /* Airbase Id Number */
     ac_name = (String)                /* Aircraft Name, Eg. M21 */
     ac_role = (String)               /* Aircraft Role, Eg. A,C,S,etc */
     oct_quantity = (Numerical)        /* Actual Quantity of Strike
                                          Aircraft on Airbase */
     atk_quantity = (Numerical)        /* Actual Quantity of Attack
                                          Aircraft on Airbase */
     aug_quantity = (Numerical)        /* Actual Quantity of Attack
                                          Aircraft of Same Type on
                                          Augmentation base */

     req_quantity = (Numerical)        /* Required Quantity of Strike
                                          Aircraft Needed on Airbase */
  PARENTS:
     airbase
     aircraft

stk_ac_on_ab
     ab_id = (Numerical)               /* Airbase Id Number */
     ac_name = (String)                /* Aircraft Name, Eg. M21 */
     ac_role = (String)               /* Aircraft Role, Eg. A,C,S,etc */
     act_quantity = (Numerical)        /* Actual Quantity of Strike
                                          Aircraft on Airbase */
     req_quantity = (Numerical)        /* Required Quantity of Strike
                                          Aircraft Needed on Airbase */
  PARENTS:
     airbase
     aircraft
```

B.3 Rule Editor

The rule editor is the most complex utility in the Nexpert developmental environment. This is due to the numerous tasks that can be accomplished. Thus the file created with this editor is the longest and hardest to make legible. The *general* steps for creating a rule include

1. Start the rule editor
2. Select the new rule option from the display
3. Enter the rule's hypothesis
4. Enter the rule's condition(s)
5. Enter the rules's actions(s)
6. Select the save rule option from menu

If you utilize any of the database options, another screen will prompt you for information such as database type and database/knowledge base conversion parameters. The database utility screen allows you to choose from a list of available database formats. For this research I used the Nexpert database format or NXPDB. One of the hardest things to understand when first using the database window is how to match a database table and its attributes with a knowledge base's class and its properties. A tuple in a relational database table maps to an knowledge base object through the use of a *name filter*. This filter has the following format:

"root1"!field1!"root2"!field2!

root1 and *root2* are simple character strings that will be concatenated to the actual database attributes that have the name *field1* and *field2*. Let's look at an example. Say you have the database table, *rd_airbase*, with the attributes, *ab_id* and *status*. Knowing this, you create a KB class called *airbase* and you give it the two properties, *ab_id* and *status*. Below is a table with sample data.

ab_id	*status*
23	1.00
24	0.50
48	0.25

The quest'⁻ⁿ now is how do you get these two structures together. First you must note that every object in a knowledge base must be unique. Thus you will need to use the ab_id as a unique qualifier. However you can not have an object that starts with a number. **(This is not explained in the Nexpert manual.)** In order to overcome this small problem you use the *root* strings to make the objects more understandable. You then create the name filter, **"ab"!ab_id!**, and link the objects to the KB class *aircraft*. Database attributes are then transferred by mapping them to KB properties. The final result of the transfer is the creation of 3 objects within the KB class *aircraft*. The names of those three object are ab23, ab24, and ab48. For more information look at the Database Links Chapter in the Nexpert manual.

The *print to file* option for the rule editor does work! After downloading the rules to a file, I then added enough tabs and comments to help other understand what each rule was responsible for in the knowledge base. The actual description of the keywords within the rules can be found in the Rules and Database Links chapters of the Nexpert manual. The following is the rules section of the nuclear strike aircraft replacement knowledge base.

```
/***********************************************************************
        Name:  Rules for the Nuclear Strike Aircraft Replacement KB
      Author:  Capt H. Dallas Harken
        Date:  1 October 1989
     Version:  1.0

    Software:  Nexpert (IBM AT) Version 1.0

 Description:  This file contains all knowledge base rule for the
               nuclear strike aircraft replacement KB.  The format
               is:

                  If
                      CONDITION(S)

                  Then HYPOTHESIS
                  is confirmed.

                      ACTION(S)

               This file was created using the SAVE TO FILE option
```

```
                within the Rule Editor.
*******************************************************************/

RULES:

/*******************************************************************
 Hypothesis:  bring_atk_ac_from_aug_ab_all

 Conditions:  This rule is fired if
              1. The required number of strike aircraft is greater
                 than the actual number of aircraft on the airbase
              2. The number of attack aircraft at the airbase is 0
              3. The retrieval of the actual number of attack
                 aircraft at the augmentation base is successful
              4. The number of needed attack aircraft cannot be
                 completely satisfied by the aircraft located at the
                 augmentation base
              5. The number of attack aircraft at the augmentation
                 base is greater than 0

    Actions:  1. Assign ab_id to temporary object, augm_temp
              2. Assign ac_name to temp. object, augm_temp
              3. Set number of attack aircraft left on augmentation
                 base to 0
              4. Update Database (see Metaslot for object, augm_temp)
              5. Reset Rule for Re-Rolling Aircraft

       NOTE: Attack and strike aircraft must be of the same type,
             eg. M21A <--> M21S
*******************************************************************/

Rule 1
If
     <atk_ac_on_ab>.atk_quantity+(<atk_ac_on_ab>.act_quantity-
     <atk_ac_on_ab>.req_quantity) is less than 0.00
And  <|atk_ac_on_ab|>.atk_quantity is precisely equal to 0.00
And  Retrieve aug.db @NXPDB;@NAME="ab_"!ab_id!;@PROPS=aug_quantity;
     @FIELDS=quantity;@ATOMS=<|atk_ac_on_ab|>.aug_quantity;
And  <atk_ac_on_ab>.aug_quantity+(<atk_ac_on_ab>.act_quantity-
     <atk_ac_on_ab>.req_quantity) is less than 0.00
And  <|atk_ac_on_ab|>.aug_quantity is greater than 0.00

Then  bring_atk_ac_from_aug_ab_all
is confirmed.

And  aug_all_flag is set to TRUE
And  <|atk_ac_on_ab|>.ab_id is assigned to augm_temp.id
And  <|atk_ac_on_ab|>.ac_name is assigned to augm_temp.ac
And  0 is assigned to augm_temp.left
And  <|atk_ac_on_ab|>.aug_quantity is assigned to augm_temp.diff
And  Reset rerole_atk_ac_from_same_base_all
And  Reset rerole_atk_ac_from_same_base_some

/*******************************************************************
 Hypothesis:  bring_atk_ac_from_aug_ab_some

 Conditions:  This rule is fired if
              1. The required number of strike aircraft is greater
                 than the actual number of aircraft on the airbase
              2. The number of attack aircraft at the airbase is 0
              3. The retrieval of the actual number of attack
                 aircraft at the augmentation base is successful
              4. The number of needed attack aircraft can be
                 completely satisfied by the aircraft located at the
                 augmentation base

    Actions:  1. Assign ab_id to temporary object, augm_temp
              2. Assign ac_name to temp. object, augm_temp
              3. Set number of attack aircraft left on augmentation
                 base to quantity available - quantity needed
              4. Update Database (see Metaslot for object, augm_temp)
              5. Reset Rule for Re-Rolling Aircraft

       NOTE: Attack and strike aircraft must be of the same type,
             eg. M21A <--> M21S
*******************************************************************/

Rule 2
If
     <atk_ac_on_ab>.atk_quantity+(<atk_ac_on_ab>.act_quantity-
     <atk_ac_on_ab>.req_quantity) is less than 0.00
And  <|atk_ac_on_ab|>.atk_quantity is precisely equal to 0.00
And  Retrieve aug.db @NXPDB;@NAME="ab_"!ab_id!;@PROPS=aug_quantity;
     @FIELDS=quantity;@ATOMS=<|atk_ac_on_ab|>.aug_quantity;
And  <atk_ac_on_ab>.aug_quantity+(<atk_ac_on_ab>.act_quantity-
     <atk_ac_on_ab>.req_quantity) is greater than or equal to 0.00

Then  bring_atk_ac_from_aug_ab_some
is confirmed.

And  aug_all_flag is set to FALSE
And  <|atk_ac_on_ab|>.ab_id is assigned to augm_temp.id
And  <|atk_ac_on_ab|>.ac_name is assigned to augm_temp.ac
And  <atk_ac_on_ab>.aug_quantity-abs(<atk_ac_on_ab>.act_quantity-
     <atk_ac_on_ab>.req_quantity) is assigned to augm_temp.left
And  abs(<atk_ac_on_ab>.act_quantity-<atk_ac_on_ab>.req_quantity)
     is assigned to augm_temp.diff
And  Reset rerole_atk_ac_from_same_base_all
And  Reset rerole_atk_ac_from_same_base_some

/*******************************************************************
 Hypothesis:  data_loaded
 Conditions:  This rule is fired if
              1. The retrieval of the required strike aircraft data
                 is successful
              2. The retrieval of the actual strike aircraft data
                 is successful

    Actions:  None  (See Rule Contexts)
*********************************************************************/

Rule 3
If
     Retrieve rstrike.db @NXPDB;@ADD;@NAME="ab_"!ab_id!;
     @CREATE=|stk_ac_on_ab|;@PROPS=ab_id,ac_name,ac_role,req_quantity;
     @FIELDS=ab_id,ac_name,ac_role,quantity;
And  Retrieve astrike.db @NXPDB;@NAME="ab_"!ab_id!;@PROPS=act_quantity;
     @FIELDS=quantity;@ATOMS=<|stk_ac_on_ab|>.act_quantity;

Then  data_loaded
is confirmed.

*********************************************************************
 Hypothesis:  low_on_stk_ac

 Conditions:  This rule is fired if
              1. The actual number of strike aircraft is less than the
                 required number

    Actions:  1. Retrieve the actual number of attack aircraft on the
                 airbase of the same type as the strike aircraft
              2. Fire rules responsible for re-roling aircraft

       Note:  Nexpert does not allow any type of object or property
              on the right-hand side of a comparison only numeric
              values, ie.

                     a < b     is illegal
                   (a b) < 0   is legal

*********************************************************************/

Rule 4
If
     <stk_ac_on_ab>.act_quantity-<stk_ac_on_ab>.req_quantity
      is less than 0.00

Then  low_on_stk_ac
is confirmed.

And  Create Object <|stk_ac_on_ab|> |atk_ac_on_ab|
And  Retrieve from aattack.db @NXPDB;@NAME="ab_"!ab_id!;
```

```
     @PROPS=atk_quantity;@FIELDS=quantity;@ATOMS=
     <|atk_ac_on_ab|>.atk_quantity;
And  rerole_atk_ac_from_same_base_all is assigned to
     rerole_atk_ac_from_same_base_all
And  rerole_atk_ac_from_same_base_some is assigned to
     rerole_atk_ac_from_same_base_some

/*********************************************************************
 Hypothesis:  rerole_atk_ac_from_aug_ab_all

 Conditions:  This rule is fired if
              1. The required number of strike aircraft is greater
                 than the actual number of aircraft on the airbase
              2. The number of needed attack aircraft cannot be
                 completely satisfied by the aircraft located at the
                 airbase base
              3. The number of attack aircraft at the airbase
                 base is greater than 0

    Actions:  1. Let KB know all attack aircraft on airbase
                 are being re-roled
              2. Assign ab_id to temporary object, res_temp
              3. Assign ac_name to temp. object, res_temp
              4. Set number of attack aircraft left on airbase
                 base to 0
              5. Update Database (see Metaslot for object, res_temp)

       NOTE: Attack and strike aircraft must be of the same type,
             eg. M21A <--> M21S
*********************************************************************/

Rule 5
If
     <atk_ac_on_ab>.act_quantity-<atk_ac_on_ab>.req_quantity
      is less than 0.00
And  <atk_ac_on_ab>.atk_quantity+(<atk_ac_on_ab>.act_quantity-
     <atk_ac_on_ab>.req_quantity) is less than 0.00
And  <|atk_ac_on_ab|>.atk_quantity is greater than 0.00

Then  rerole_atk_ac_from_same_base_all
is confirmed.

And  rerole_all_flag is set to TRUE
And  <|atk_ac_on_ab|>.ab_id is assigned to res_temp.id
And  <|atk_ac_on_ab|>.ac_name is assigned to res_temp.ac
And  0 is assigned to res_temp.left
And  <|atk_ac_on_ab|>.atk_quantity is assigned to res_temp.diff

/*********************************************************************
 Hypothesis:  rerole_atk_ac_from_aug_ab_all
```

```
Conditions:  This rule is fired if
             1. The required number of strike aircraft is greater
                than the actual number of aircraft on the airbase
             2. The number of needed attack aircraft can be
                completely satisfied by the aircraft located at the
                airbase base

   Actions:  1. Let KB know not all attack aircraft on airbase
                are being re-roled
             1. Assign ab_id to temporary object, res_temp
             2. Assign ac_name to temp. object, res_temp
             3. Set number of attack aircraft left on airbase
                base to quantity available - quantity needed
             4. Update Database (see Metaslot for object, res_temp)

      NOTE: Attack and strike aircraft must be of the same type,
            eg. M21A <--> M21S
**********************************************************************/

Rule 6
If
     <atk_ac_on_ab>.act_quantity-<atk_ac_on_ab>.req_quantity
     is less than 0.00
And  <atk_ac_on_ab>.atk_quantity+(<atk_ac_on_ab>.act_quantity-
     <atk_ac_on_ab>.req_quantity) is greater than or equal to 0.00

Then  rerole_atk_ac_from_same_base_some
is confirmed.

And  rerole_all_flag is set to FALSE
And  <|atk_ac_on_ab|>.ac_name is assigned to res_temp.ac
And  <|atk_ac_on_ab|>.ab_id is assigned to res_temp.id
And  <atk_ac_on_ab>.atk_quantity-abs(<atk_ac_on_ab>.act_quantity-
     <atk_ac_on_ab>.req_quantity) is assigned to res_temp.left
And  abs(<atk_ac_on_ab>.act_quantity-<atk_ac_on_ab>.req_quantity)
     is assigned to res_temp.diff
```

B.4 The Object Editor

The manual creation of objects is not generally needed when creating a knowledge base. Most objects are created dynamically when reading data from the database. However, there are a few objects such as flags and holding areas that can be entered by hand. The procedure for creating objects is exactly like that of creating classes. You should note that the hypothesis of a rule is also an object but it is constructed automatically by the rule editor. The principal reason for using the object editor is that it allows you to add, delete, and modify the "order of sources" and "if change" actions of the objects. These actions are known as *metaslots*.

The "order of sources" action for an object allows you to determine an object's property values at anytime during a knowledge session. The most valuable action used is *InitValue*. This action assigns a value to the object during the startup of a knowledge session. The action *RunTimeValue* provides a means of changing an objects values while in the middle of knowledge session. By setting the value of an object to *unknown* (using the *reset* command), the *RunTimeValue* will be assigned to that object if it is ever evaluated by the inference engine.

The "if change" actions is another means of controlling the knowledge session outside of a rule's action set. One or more commands can be executed by simply changing the value of an object. This is most commonly implemented by creating a boolean object and changing its value from false to true. Actions associated with this metaslot are executed in the order in which they were entered.

You must also use the object editor to change the inference category of a hypothesis. The use of inference categories was explained in my knowledge base design chapters. The default inference category for a hypothesis is one. Increasing this number increases a rules priority within the inference engine.

Below is the output from the object editor using the *print* option:

```
/***********************************************************************
       Name:  Objects for the Nuclear Strike Aircraft Replacement KB
     Author:  Capt H. Dallas Harken
       Date:  1 October 1989
    Version:  1.0

   Software:  Nexpert (IBM AT) Version 1.0

Description:  This file contains all KB objects, including the
              hypotheses for each rule.  Special attention should
              be given to the ORDER OF SOURCES, IF CHANGES ACTIONS,
              and INFERENCE CATEGORY sections of each object.  If
              these section do not exist assume the default values
              are used.

       NOTE:  aug_temp and res_temp are the key objects used
              in updating the database.

              This file was created using the PRINT option
              within the Rule Editor.
**********************************************************************/

OBJECTS:

aug_all_flag                                /* Boolean Flag for determining
                                               if all planes at augmentation
                                                 base are moved */
  PROPERTIES:
     Value = (Boolean)

augm_temp                                 /* Object responsible for updating
                                             augmentation base inventory and
                                               aircraft movement log file */
  PROPERTIES:
     ac = (String)
     diff = (Numerical)
        ORDER OF SOURCES:
           InitValue 0.000000
           RunTimeValue 0.000000
        IF CHANGE ACTIONS:
           Do augm_temp.left "ab_"\augm_temp.id\.aug_quantity
           Do augm_temp.diff "ab_"\augm_temp.id\.atk_quantity
           CreateObject \augm_temp.ac\ |atk_ac_on_ab|
           DeleteObject \augm_temp.ac\ |atk_ac_on_ab|
           Do augm_temp.left \augm_temp.ac\.aug_quantity
           Write aug.db @NXPDB;@NAME=!ac_name!;@PROPS=aug_quantity;
             @FIELDS=quantity;
           DeleteObject \augm_temp.ac\
           Write augm.db @NXPDB;@ADD;@NAME=!ab_id!;@PROPS=id,ac,diff;
             @FIELDS=ab_id,ac_name,quantity;@ATOMS=augm_temp
           Do abs("ab_"\augm_temp.id\.act_quantity+"ab_"
             \augm_temp.id\.atk_quantity-"ab_"\augm_temp.
             req_quantity) augm_temp.left
           Execute augm.frm @FRM;@WAIT;
           Reset augm_temp.diff
           Reset bring_atk_ac_from_aug_ab_some
     id = Unknown (Numerical)
     left = Unknown (Numerical)

Bring_atk_ac_from_aug_ab_all             /* Hypothesis */
  PROPERTIES:
     Value = (Boolean)

bring_atk_ac_from_aug_ab_some          /* Hypothesis */
  PROPERTIES:
     Value = (Boolean)

data_loaded                            /* Hypothesis */
  PROPERTIES:
     Value = (Boolean)

low_on_stk_ac                          /* Hypothesis */
  PROPERTIES:
     Value = (Boolean)

rerole_all_flag                        /* Hypothesis */
  PROPERTIES:
     Value = (Boolean)

rerole_atk_ac_from_same_base_all       /* Hypothesis */
  PROPERTIES:
     Value = (Boolean)
     INFERENCE CATEGORY: 3

rerole_atk_ac_from_same_base_some      /* Hypothesis */
  PROPERTIES:
     Value = (Boolean)
     INFERENCE CATEGORY: 3

res_temp                             /* Object responsible for updating
                                         airbase inventory and
                                        aircraft re-role log file */
  PROPERTIES:
     ac = Unknown (String)
     diff = 0.00 (Numerical)
        ORDER OF SOURCES:
           InitValue 0.000000
           RunTimeValue 0.000000
        IF CHANGE ACTIONS:
           Do res_temp.left "ab_"\res_temp.id\.atk_quantity
           Do "ab_"\res_temp.id\.act_quantity+res_temp.dif
              "ab_"\res_temp.id\.act_quantity
           Write astrike.db @NXPDB;@NAME="ab_"!ab_id!;@PROPS=
              act_quantity;@FIELDS=quantity;@ATOMS="ab_"
              \res_temp.id\
           Write aattack.db @NXPDB;@NAME="ab_"!ab_id!;@PROPS=
              atk_quantity;@FIELDS=quantity;@ATOMS="ab_"
              \res_temp.id\
           Write rerole.db @NXPDB;@ADD;@NAME=!ab_id!;@PROPS
              =id,ac,diff;@FIELDS=ab_id,ac_name,quantity;
              @ATOMS=res_temp
           Do abs("ab_"\res_temp.id\.act_quantity-"ab_"
                \res_temp.id\.req_quantity)res_temp.left
            Execute rerole.frm @FRM;@WAIT;
            Reset res_temp.diff
            Reset rerole_atk_ac_from_same_base_some
    id = Unknown (Numerical)
    left = Unknown (Numerical)
```

B.5 The Context Editor

The context editor is responsible for determining which rules will be investigated by the inference engine after the current rule is evaluated. Using the *Propagate When True (PWT)* strategy any rule placed in context with the current rule will be placed on the system's agenda if and only if the current rule's hypothesis evaluates to true.

The context editor modifies the relationship between rules by linking their respective hypotheses together. The editor lists a rule's hypothesis and then allows you to add or delete other hypotheses to the on shown. It is possible to place a rule in context with itself. This allows you to create a "loop" within your knowledge base, and can be used to find the largest or smallest value of numerous objects within a class.

Once again you have to use the *print* option within the editor to get a hardcopy of your data. The following are the contexts used in the nuclear strike aircraft replacement knowledge base.

```
/*********************************************************************
       Name:  Rule Contexts for the Nuclear Strike Aircraft
              Replacement KB
     Author:  Capt H. Dallas Harken
       Date:  1 October 1989
    Version:  1.0

   Software:  Nexpert (IBM AT) Version 1.0

Description:  This file contains the rule contexts for the nuclear
              strike aircraft replacement knowledge base.  The top
              most hypothesis is assumed to be that of the current
              rule.  Thus all indented hypotheses below it are those
```

```
               in context with the current rule.

               Those rules with no hypotheses directly below them have
               no contextual relationships.

               This file was created using the PRINT option
               within the Context Editor.
**********************************************************************/

CONTEXTS:

bring_atk_ac_from_aug_ab_all

bring_atk_ac_from_aug_ab_all

rerole_atk_ac_from_same_base_all

rerole_atk_ac_from_same_base_some

bring_atk_ac_from_aug_ab_some

bring_atk_ac_from_aug_ab_some
   rerole_atk_ac_from_same_base_all
   rerole_atk_ac_from_same_base_some

data_loaded
   low_on_stk_ac

low_on_stk_ac

rerole_atk_ac_from_same_base_all

rerole_atk_ac_from_same_base_some
```

B.6 The Property Editor

The property editor is rarely used since property types are automatically asked for when a property is created through the use of the other editors. However if you need to change the property type this editor is as functional as the rest.

The property editor also provides a means for printing all properties and their types. This function is not really necessary since the output from the object and class editors also show these properties and types. Below is the output of the property editor provided for the sake of completeness.

```
/*****************************************************************
        Name:  Object Properties for the Nuclear Strike Aircraft
               Replacement KB
      Author:  Capt H. Dallas Harken
        Date:  1 October 1989
     Version:  1.0

    Software:  Nexpert (IBM AT) Version 1.0

 Description:  This file contains the object properties and their types
               for  the nuclear strike aircraft replacement knowledge
               base.  These values can also be found in the Object
               and Class files.

               This file was created using the PRINT option
               within the Property Editor.
*****************************************************************/

PROPERTIES:  /* Descriptions can be found in the Object and Class
                Files */

ab_id (Numerical)

ac_name (String)

ac_role (String)

act_quantity (Numerical)

atk_quantity (Numerical)

aug_quantity (Numerical)

diff (Numerical)

id (Numerical)

left (Numerical)

req_quantity (Numerical)

Value (Special)                    /* Default Value for Hypotheses */
```

B.7 The Forms Input Utility

The forms input utility allows users to control a knowledge session the use of a command script. This script can prompt the user, load the knowledge base, start the knowledge base by suggesting a hypothesis, and report the actions of the inference engine to the user's screen. This is as close to a procedural language as Nexpert gets. It even provides a means for commenting your command files; **but it has been removed from later versions due to conflicts with the runtime version of the knowledge bases**. Since I used version 1.0 of Nexpert I was able to make use of this utility. I used the command scripts to start the knowledge session and report aircraft movement and re-roling operations. An alternative solution for future efforts might be to make use of Nexpert's external interface with procedural languages such as C or Fortran. This would allow the programmer to tailor the delivery environment to the user's specific requirements.

The *report forms* for the knowledge bases are displayed by using the *execute* command within a rules action set or an object's "if change" metaslot. See the Nexpert manual for more details.

The form below is used to start the nuclear strike aircraft replacement knowledge base.

```
{THIS IS A COMMENT}

{**********************************************************************}
{    Filename: strike.frm                                              }
{      Author: Capt H. Dallas Harken                                   }
{    Ver/Date: 1.0/3 Aug 1989                                          }
{                                                                      }
{ Description: Startup form for Nuclear Strike Aircraft Replacement    }
{              Knowledge Base                                          }
{**********************************************************************}

#evaluation(OFF)
#toggle(Transcript)                           { Turn off Transcript Window }
#beep()
#caption(Maintain Strike Aircraft)
#remove_scroll()
```

```
#remove_menu()
#fontsize("24,0")
#fontcolor("RED")
#ctext("AI Demonstration for Maintaining Nuclear Strike Aircraft")
#blankspace("LINES_1")
#fontsize("12,8")
#fontcolor("BLACK")
#ltext("The following Demo will demonstrate the use of the *b[Nexpert]")
#ltext("*b[Expert System Shell] for maintaining the correct number of")
#ltext("nuclear strike aircraft at specified bases.")
#blankspace("LINES_2")
#ctext("Click on START to continue.")    {Use Mouse}
#blankspace("LINES_1")
#button("START", OK, CENTER)

{ The Real Work Begins Here }

#loadkb(strike.kb)                       { Load Knowledge Base }
#suggest(data_loaded)
#knowcess()                              { Start Knowledge Session }
```

This second form reports aircraft re-roling operations.

```
{*******************************************************************}
{        File: rerole.frm                                           }
{      Author: Capt Dallas Harken                                   }
{    Ver/Date: 1.0/3 Aug 89                                         }
{ Description: This is a report form used to show aircraft re-roling }
{               operations                                          }
{*******************************************************************}

#evaluation(ON)
#remove_menu()
#remove_scroll()
#caption(rerole.frm)
#beep()

#if(rerole_all_flag == False)             { Flag Set by Knowledge Base }

#fontcolor("RED")
#fontsize("24,0")
#ctext("Re-role All Aircraft From Same Base")
#blankspace("LINES_2")
#fontcolor("BLACK")
#fontsize("12,8")
#ltext("Attack Aircraft on the following base are being re-roled")
#ltext("to Nuclear Strike Aircraft.  The total number of aircraft")
#ltext("needed will be re-roled.")
```

```
#blankspace("LINES_2")
#ltext(" AirBase ID:  \res_temp.id\" )
#ltext(" AirCraft Name:  \res_temp.ac\" )
#ltext(" Number of Aircraft Re-roled:  \res_temp.diff\" )

#else

#fontcolor("RED")
#fontsize("24,0")
#ctext("Re-role Some Aircraft From The Same Base")
#blankspace("LINES_2")
#fontcolor("BLACK")
#fontsize("12,8")
#ltext("Attack Aircraft on the following base are being re-roled")
#ltext("to Nuclear Strike Aircraft.  The total number of aircraft")
#ltext("needed exceeds the number of attack aircraft available on base.")
#ltext("The difference will be brought in from the augmentation base.")
#blankspace("LINES_2")
#ltext(" AirBase ID:  \res_temp.id\" )
#ltext(" AirCraft Name:  \res_temp.ac\" )
#ltext(" Number of Aircraft Re-roled:  \res_temp.diff\" )
#ltext(" Number of Aircraft Still Needed:  \res_temp.left\")

#endif

#blankspace("LINES_2")
#button("Continue",OK, CENTER)            { Use Mouse }
#evaluation(OFF)
#knowless()                              { Continue Knowledge Session }
```

The final form in the nuclear strike aircraft replacement knowledge base reports aircraft movements.

```
{*******************************************************************}
{        File: augm.frm                                             }
{      Author: Capt Dallas Harken                                   }
{    Ver/Date: 1.0/3 Aug 89                                         }
{ Description: This is a report form used to show aircraft movement }
{               operations                                          }
{*******************************************************************}

#evaluation(ON)
#remove_menu()
#remove_scroll()
#caption(mvaug.frm)
#beep()
```

```
#if(aug_all_flag == False)                { Flag Set By Knowledge Base }

#fontcolor("RED")
#fontsize("24,0")
#ctext("Move All Aircraft From Augmentation Base")
#blankspace("LINES_2")
#fontcolor("BLACK")
#fontsize("12,8")
#ltext("Attack Aircraft from the Augmentation base are being trans-")
#ltext("ferred to the following base for rerole to Nuclear Strike")
#ltext("Aircraft.  The total number of aircraft needed will be")
#ltext("transferred.")
#blankspace("LINES_2")
#ltext(" AirBase ID:  \augm_temp.id\" )
#ltext(" AirCraft Name:  \augm_temp.ac\" )
#ltext(" Number of Aircraft Moved:  \augm_temp.diff\" )

#else

#fontcolor("RED")
#fontsize("24,0")
#ctext("Move Some Aircraft From Augmentation Base")
#blankspace("LINES_2")
#fontcolor("BLACK")
#fontsize("12,8")
#ltext("Attack Aircraft from the Augmentation base are being trans-")
#ltext("ferred to the following base for rerole to Nuclear Strike")
#ltext("Aircraft. The total number of aircraft needed exceeds the")
#ltext("number of aircraft available at the augmentation base.")
#ltext("The difference will have to be brought in from other bases.")
#blankspace("LINES_2")
#ltext(" AirBase ID:  \augm_temp.id\" )
#ltext(" AirCraft Name:  \augm_temp.ac\" )
#ltext(" Number of Aircraft Moved:  \augm_temp.diff\" )
#ltext(" Number of Aircraft Still Needed:  \augm_temp.left\")

#endif

#blankspace("LINES_2")
#button("Continue",OK, CENTER)            { Use Mouse }
#evaluation(OFF)
#knowcess()                               { Continue Knowledge Session }
```

B.8 Summary

There not much left to say for documenting a knowledge base's code. My final recommendation however is, if you have any questions, **Look it up in the manual!** Good-Luck.

Bibliography

1. Astrahan, M.P., Blasgen, M.W., Chamberlain, D.D., Eswaren, J.N., and others "System R: Relational Approach to Database Management," *ACM Transactions on Database Systems 1*, 97–137 (1976).

2. Barr, A. and Feigenbaum, E.A.; Eds. *The Handbook of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., Vol 1: 1981.

3. Brachman, R.J., Gilbert, V.P., and Levesque, H.J. "An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of Krypton," Artificial Intelligence and Databases, Morgan Kaufmann Publishers Inc., 293–300 (1989).

4. Boar, Bernard H. *Application Prototyping*. John Wiley & Sons, New York, 1984.

5. Bodie, M.L. and Mylopoulos, J.; Eds. *On Knowledge Base Management Systems*, Springer-Verlag New York Inc., 1986.

6. Brooks, Capt Michael. *Developing a Database Management System and Air Simulation Software for a Theater War Exercise (ADA189681)*. Master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1987. AFIT/GCS/ENG/87D-6.

7. Brodie, M.L. "Future Intelligent Information Systems: AI and Database Technologies Working Together," *Artificial Intelligence and Databases*, Morgan Kaufmann Publishers, Inc., 1989.

8. Carr, Michael A. and others *Building Knowledge Systems*, McGraw-Hill, 1989.

9. Korth, H.F. and Silberschatz, A. *Database System Concepts*, McGraw-Hill, 1986.

10. Kross, Capt Mark S. *Developing New User Interfaces for the Theater War Exercise (ADA189744)*. Master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1987. AFIT/GCS/ENG/87D-19.

11. Levesque, H.J. and Brachman, R.J. "Knowledge Level Interfaces to Information Systems," *On Knowledge Base Management Systems*, Springer-Verlag New York Inc., 13–34 (1986).

12. Manola, F. and Brodie M.L. "On Knowledge–Based System Architectures," *On Knowledge Base Management Systems*, Springer-Verlag New York Inc., 35–54 (1986).

13. Mylopoulos, J. and Brodie, M.; Eds. *Artificial Intelligence and Databases*, Morgan Kaufmann Publishers, Inc., 1989.

14. Napheys, B. and Herkimer, D. "A Look at Loosely-Coupled Database Systems," *Proceeding of the Second International Conference on Expert Database Systems*, 107–115 (April 1988).

15. Neuron Data Inc. *Nexpert Object Fundamentals*, Palo Alto, CA, Version 1.0 (IBM-AT), 1988.

16. Quick, Capt Darrell A. *Adding Map-Based Graphics to the Theater War Exercise (ADA205902)*. Master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1988. AFIT/GCS/ENG/88D-16.

17. Koussopoulos, N. and Mylopoulos J. "Using Semantic Networks for Database Management," *Artificial Intelligence and Databases*, Morgan Kaufmann Publishers, Inc., 112–137 (1989).

18. Shortliffe, E.H. *Computer–Based Medical Consultations: MYCIN*, Elsevier New York, 1976.

19. Somsel, J. "NEXPERT Object and Humble: Object-Based Shells," *AI Expert*, Nov 1987.

20. Tanimoto, S.L *The Elements of Artificial Intelligence*, Computer Science Press, 1987.

21. Turing, A.M. "Computing Machinery and Intelligence," *Computers and Thought*, McGraw Hill, 11–35 (1963).

22. *Theater Warfare Exercise Handbook.* Air Force Wargaming Center, Maxwell AFB, AL, 1988. Unpublished Manual.

23. *TWX Soviet / Warsaw Pact: Operation Red Lightning.* Air Force Air War College, Maxwell AFB, AL, 1988.

24. Van Horn, Michael *Understanding Expert Systems.* Bantam Books, 1986.

25. Department of Defense. *USAF CoS Constant Readiness Tasking.* DOD Directive, Item 6. Washington: Government Printing Office, 4 August 1976.

26. Wilcox, Capt Kenneth R. *Extending the User Interface for The Theater War Exercise (ADA202726).* Master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1988. AFIT/GCS/ENG/88D-24.

Vita

Captain Harold "Dallas" Harken III was born on 8 April 1963 at Travis Air Force Base, California, to Mr. and Mrs H. D. Harken, Jr. He graduated with honors from Middleton High School in 1981. He then attended Clemson University on an Air Force ROTC scholarship and received a Bachelor of Science in Computer Engineering in 1985. Following graduation, 2Lt Harken was assigned to the 7th Communications Group at the Pentagon, where he served as a UNIX system administrator and programmer. In 1988 he was accepted by the Air Force Institute of Technology as a masters student in the School of Engineering. After graduation in December (1989), Capt Harken will be assigned to the Air Force Wargaming Center at Maxwell AFB, Alabama.

Permanent address: *1433 Birthright Street*
Charleston, South Carolina
29407

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/ENG/89D-7	5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433-6583		7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Air Force Wargaming Center	8b. OFFICE SYMBOL (If applicable) AUCADRE/WG	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Maxwell AFB, AL 36112-5532		10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
An Expert System for Automating Nuclear Strike Aircraft Replacement, Aircraft Beddown, and Logistics Movement for the Theater Warfare Exercise (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
Harold D. Harken III, B.S., Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM ______ TO ______	1989 December	136

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	5		Wargaming, Expert Systems, Database, Prototyping

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Advisor: Mark A. Roth, Major, USAF
Associate Professor of Computer Systems

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED [✓ SAME AS RPT] ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Mark A. Roth, Major, USAF	22b. TELEPHONE (Include Area Code) (513) 255-3576	22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

The Theater War Exercise (TWX) is a five day, two sided, theater level, air-power employment decision making exercise. The decisions required are typical of those that an air component commander and staff would make. TWX is a two-sided game where the blue team is played by a student seminar and the red team is played by one or more dedicated Air Force Wargaming Center personnel who attempt to provide a realistic red opponent.

Personnel at the Air Force Wargaming Center determined that too much time was required for a red player to render an effective game. Also noted was the divergent background of the red players made it difficult to play a normalized game during multiple seminars. The goal of this thesis was to evaluate existing software programs, determine which would best serve as a platform for automating the red player, design a system to that effect, and implement it.

It was determined that an integration of artificial intelligence and relational database management systems would provide a flexible, innovative, and cost-effective approach for automation. Nexpert Object, an expert system shell by Neuron Data, was chosen as the software platform.

An object-oriented approach was used to determine the necessary structures for automating the planning section of TWX. This included the replacement of nuclear strike aircraft, the beddown of aircraft from an augmentation base, and the resolution of logistic shortfalls at each airbase due to attrition and movement of aircraft

The creation of three knowledge bases resulted from the design phase using application prototyping, which facilitated the need for constant changes to the rules in order to present a system that acted in accordance with the desire of the red players. This new series of programs provided a means of lessening the red player's time involved with simplistic, but time-consuming work and allowed them to increase their time on the sections dealing with target selection and prioritization.